MASTER'S DEGREE THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Analyzing SPSA approaches to solve the non-linear non-differentiable problems arising in the assisted calibration of traffic simulation models

Author: Xavier Ros Roca

Advisor: Lídia Montero Mercadé

Advisor: Jaume Barceló Bugeda

Department: Departament d'Estadística i Investigació Operativa

University: Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Statistics and Operations Research
Master's Degree Thesis

Analyzing SPSA approaches to solve the non-linear non-differentiable problems arising in the assisted calibration of traffic simulation models

Xavier Ros Roca

Advisor: Lídia Montero Mercadé
Co-advisor: Jaume Barceló Bugeda

Departament d'Estadística i Investigació Operativa

Vísteme despacio,
que tengo prisa

A la Marta Grau Montsalvatge,
benvinguda al món
(27/12/2016)

Abstract

Mathematical and simulation models of systems lay at the core of decision support systems, and their role become more critical as more complex is the system object of decision. The decision process usually encompasses the optimization of some utility function that evaluates the performance indicators that measure the impacts of the decisions. An increasing difficulty directly related to the complexity of the system arises when the associated function to be optimized is a not analytical, non-differentiable, non-linear function which can only be evaluated by simulation. Simulation-Optimization techniques are especially suited in these cases and its use is increasing in traffic models, an archetypic case of complex, dynamic systems exhibiting highly stochastic characteristics. In this approach simulation is used to evaluate the objective function, and a non-differentiable optimization technique to solve the optimization problem is used. *Simultaneous Perturbation Stochastic Approximation* (SPSA) is one of the most popular of these techniques.

This thesis analyzes, discusses and presents computational results for the application of this technique to the calibration of a traffic simulation model of a Swedish highway section. Variants of the SPSA, replacing the usual gradient approach by a combination of normalized parameters and penalized objective function, have been proposed in this study due to an exhaustive analysis of the behavior of classical SPSA where problems arose from different magnitude variables.

In this work, a varied set of Software environments have been used, combining `RStudio` for the analysis, Python and *MATLAB* for the SPSA implementation, *AIMSUN* as a Traffic Model Simulator, and `SQLite` for obtaining of simulated data and *Tableau* for visualizing data and results.

Acknowledgements

Primerament, agrair especialment l'esforç i dedicació de la Lídia i del Jaume per tirar endavant aquest treball. No ha estat fàcil compaginar-nos i estic molt agraït de l'esforç i el temps invertit.

Donar les gràcies també a la direcció del Màster i a Accenture Analytics, en especial a l'Elsa i la Paula, per les facilitats que ens han donat per poder acabar el Treball de Fi de Màster i seguir treballant. Ho feu tot més fàcil.

De forma indirecta, molta gent del meu entorn ha posat el seu gra de sorra en aquest Màster. Gent amb la que he treballat a Accenture, com el Joan, Tomás i Gabi i gent que s'ha interessat, ha ajudat, donat suport o ha fet una lectura crítica de la memòria, gràcies Albert, Anna i Marina.

Finalment a la família, que dóna suport sempre, i a bons amics que han estat aprop, la Glòria, la Paula, el Jaume, l'Adrià, la Laura, l'Arnau i molts més. Sense oblidar a nova gent, nous amics d'Accenture, l'Alberto, el Carlos, l'Anna, la Maria, l'Ana i molts més, que fan el dia a dia més lleuger.

Contents

1 Introduction

This introductory chapter highlights the importance of modeling real systems and simulation as a computer technique to experience different situations in a real system. It also establishes a model building methodology.

1.1 Modeling Real Systems

Understanding the behavior of real-life dynamics is a problem that humans pursue since ages. Many of the things around humans are complex interactions of different agents that act with a common purpose, these sets are usually called *Systems*. A *Model* of a system tries to build an accurate representation of a system in order to understand how it works, behaves and evolves with time. Models of systems, physical, human or hybrid, make part of science since its origins. For instance, Physics' laws of Dynamics are a model of how some punctual particles behave under the application of different forces.

Modeling requires abstraction, simplification and acquisition of knowledge about how it works. Lots of well designed experiments have to be done in order to collect statistics and derive some rules of conduct of the system.

Having a model of a system permits to predict the output of the system under various conditions without using the real system. The output is an aspect of interest for the modeler and it could support decision making and could answer some *what if* questions about the behavior of the system under some new conditions.

1.1.1 Methodological Framework for model building

Building a model is not an easy work. First of all, one needs to acquire knowledge of how the system works. This knowledge has to be translated to mathematical or logical relationships.

It is important to state a model building process which helps to adjust the model to the real system. This methodology begins with the system analysis and a process of knowledge acquisition. All the knowledge has to be transformed to a conceptual model of the system. According to [8], in this first step one has to distinguish three major components:

1) **Elements of Structure**: Aspects or components that are stable, or change slowly, in the time frame studied. These can be physical structures, equipment or properties of certain components.

2) **Elements of Process**: Aspects of the system that evolve during the time frame. These can be flow of material, processing of information, etc.

3) **Relationships between Structure and Process and between Processes**: Constraints for the processes induced by the structure or other processes.

All relevant components must be identified and fully described. This description is the basis of the system model and consists in specifying the transformations of the system, the different components and subsystems of the system, its boundaries and the inputs and outputs of the system. The system analysis is crucial and permits to have a conceptual model of the studied system. This conceptual model contains enough knowledge and the formulation of hypotheses on how the system works and to characterize the entities' relationships and interactions.

One difficult step is to translate the conceptual model to a computer model, through a mathematical representation via numerical algorithms, that is able to reproduce the real system. In real life, there is variability which is difficult to capture in deterministic mathematical models. Usually, one can supply it by using random variables to certain parameters of the model, opening the range of inputs to a known distributions. The computer models have to be validated and checked regularly so as to verify they approximate the real system.

Once the computer model is built, it has to be calibrated and validated. These two steps, that will be explained in detail in next Section, are essential to make the model usable for its purpose such as predicting outputs when changing some inputs of the model.

Figure 1.1 is a schema of the methodology described for a model building.

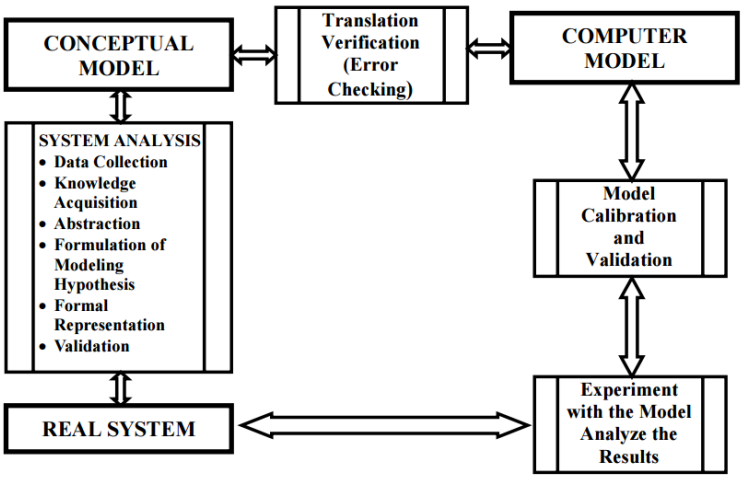


Figure 1.1: Methodological steps of the model building process. Taken from [5].

1.2 What is simulation?

Simulation is a technique used to experience different situations on a real system using a computer model. Simulation is an alternative to analytical models consisting of a technique that imitates the behavior of the real system when the modeling exercise involves to reproduce the dynamic evolution of the system and considers explicitly stochastic elements. The results provided by the computer model can be assimilated as the system output assuming some errors based on the assumptions taken when modeling. Thus, it is really important to build a model that can reply a closely similar behavior of the real system.

A simulation is a laboratory experiment made in a computer with the intention of understanding how changes in the input can affect the desired output. The validity of the answer comes directly from the validity of the model built. Therefore, what makes a model useful?

A quantification of the usability of a model is a classic Statistics statement

$$\mathrm{P}\left(|\mathit{Reality} - \mathrm{Model}| < \mathrm{d}\right) > \alpha$$

where d is the tolerable difference and α the level of assurance given by the model. These values are completely fixed by the modeler, who decides as a function of the availability and quality of data and knowledge about the system.

1.3 Calibration & Validation

As already stated, a modeler wants to have the best model to represent the system with the purpose of having the best answers to some experiments. *Calibration & Validation* are two nested steps in a model building that can transform a bad model into a useful and valid model.

This step is an iterative process that calibrates the model parameters, compares the model to the actual system behavior and uses the discrepancies between the two and the insight gained, to improve the model until the accuracy is judged to be acceptable, [5]. Figure 1.2 shows perfectly the iterative process in a diagram.

This process aims to find the best values for the model parameters and produce a valid model. Calibration uses available observed data from the real system to adjust the parameters for a particular situation and Validation contrasts the solution provided by the simulation with the adjusted parameters with another independent set of observed data.

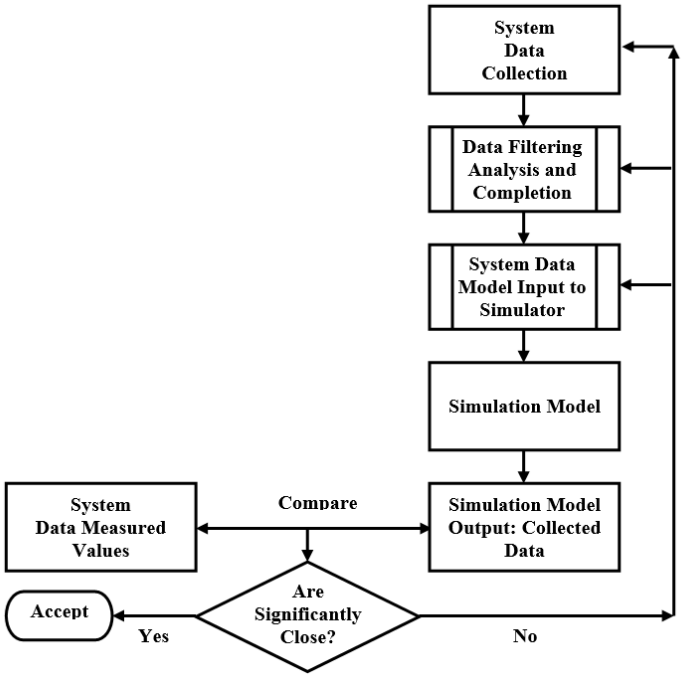


Figure 1.2: Methodological Scheme for Calibration and Validation of Simulation Models. Taken from [5].

2
Calibration Methodologies

This chapter shows four different calibration techniques. As defined in Chapter 1, model calibration is the process that adjusts model parameters in order to obtain the most accurate simulated data. A good summary of calibration methods can be found in [6] and [19].

2.1 Trial and Error

Computer models are usually mathematical models that depend on many parameters. These parameters give a wide range of possibilities and they must be adjusted to bring to the best fit of the real system.

Experience and knowledge in the field of research and some useful information about the model allow setting the parameters not completely at random. For example, if one parameter is the vehicle speed in a highway, in km/h, one knows that it has to be between 85 and 120 km/h. That pre-knowledge provides a good approximation to the best parameters combination with a clearly less computational effort. These values are set as default values and used as a first approximation and, depending on the results given, they can be slightly modified.

However, this manual choice relies on the modeler experience and judgment, and can become a good calibration but not the best, biased by some wrong hypotheses. Therefore, *Trial and Error*, as the name says, consists in trying different parameters combination since one combination gives reasonable results.

2.2 Simulation Optimization Techniques

Simulation Optimization techniques use mathematical programming in order to find the best parameters for the model. An optimization problem with the objective of minimizing an error function between real observed data of the system and simulated data extracted from the simulator can be defined.

More formally, let $\mathbf{P} = (p_1, \cdots, p_N)$ be the parameters of the model M which needs to be calibrated. Let $f_E(\mathcal{R}, \mathcal{S})$ be an error function between real observations, $\mathcal{R}$, and

the corresponding simulated data, $\mathcal{S}$. Hence, the problem can be written as follows:

$$\begin{cases} \min & f_E(\mathcal{R}, \mathcal{S}) \\ \text{s. to} & \\ & \mathbf{P} \in \Omega \subseteq \mathbb{R}^N \end{cases} \tag{2.1}$$

where Ω represents the open set of $\mathbb{R}^N$ where parameters values are feasible.

Despite the problem defined above is of the form of a classical minimizing problem, it can not be solved with the usual algorithms, such as Newton-Rhapson method, mainly for some reasons listed below:

1) These optimization problems are usually non-convex and non-linear which require high computational effort.

2) Function $f_E(\mathcal{R}, \mathcal{S})$ can not be represented analytically as a function of the parameters $\mathbf{P} = (p_1, \cdots, p_N)$. Therefore, it is neither differentiable with respect of the parameters.

3) Function $f_E(\mathcal{R}, \mathcal{S})$ is expensive in time since one has to run the simulation in order to have simulated data at each evaluation of the function.

Heuristic methods iterate moving the values of the parameters looking for the combination that reaches a minimum of $f_E(\mathcal{R}, \mathcal{S})$. They are particularly useful because they do not need many evaluations of the function in order to move to the next point. This Simulation-Based Optimization methods are iterative based in two steps as seen in Figure 2.1.

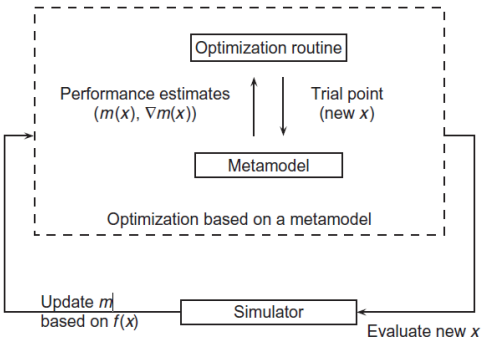


Figure 2.1: Simulation-Based Optimization Methods. Taken from [13].

Step 1 builds a metamodel which provides an analytical approximation of the objective function based on the current simulated observations. Step 2 uses this metamodel to derive a new point to simulate again by using some optimization techniques.

Since SPSA method is the one that will be used in this thesis, it will be further explained in Section 2.2.3.

2.2.1 Genetic Algorithms (GA)

Genetic Algorithms are widely used to solve large problems because they do not require gradient information for $f_E(\mathbf{P})$ with respect to parameters, they can perform the search of the optima from different starting points which eases to reach the global minima and they guarantee the convergence, [20].

Genetic Algorithms are based in biological principles, in how children inherit properties from the generation before, using genetics phenomena such as *Mutation*, *Selection* or *Replacements* in chromosomes. Hence, in the algorithm, first generation of feasible combinations of parameters is selected at random and next generations are mutations, selections and combinations of the first ones. Figure 2.2 summarises the algorithm:

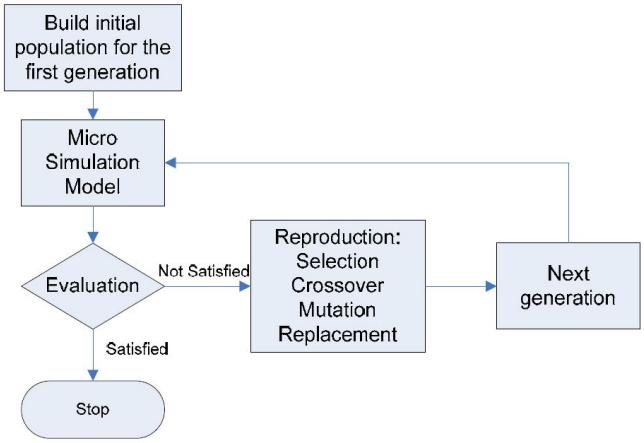


Figure 2.2: Conceptual Procedure of GA based Calibration, taken from [20].

2.2.2 Noisy Reduction Optimization (*SNOBFIT*)

This optimization method is used when there is a noise associated to the objective function, usually due to experimental errors or low-precision calculations, [11]. The variability of the objective is usually eliminated by calling many times the objective function. This procedure is not valid when large computations are needed to compute the value, as in the Simulation-Optimization problems.

SNOBFIT, which is implemented in a *MATLAB* package, is a method that solve these problems, when $f_E(\mathcal{R}, \mathcal{S})$ requires large computational times and presents variability. As explained in [11], the algorithm builds internally a stochastic interpolation of the objective function around each point, and returns a number of points whose evaluation is likely to improve these models or is expected to give better function values.

This method does not require gradients and converges in very few function evaluations in low-dimensional problems.

2.2.3 Simultaneous Perturbation Stochastic Approximation (SPSA)

The heuristic method called *Simultaneous Perturbation Stochastic Approximation* was firstly defined by *J. Spall* in [14]. The same author in 1998 wrote two articles, [15, 16], where the guidelines of the method where fully defined. This section is strongly based on these two papers.

SPSA is a very used method when the gradient of the function can not be calculated due to its complexity. Its strongest point is that it only requires two evaluations of the function instead of N in the case of finite differences approach.

As many of iterative algorithms, it starts from an initial combination of parameters:

$$\mathbf{P}_0 = (p_1^0, \cdots, p_N^0)$$

And the next point can be founded as usually, using the first order Taylor development:

$$\mathbf{P}_{k+1} = \mathbf{P}_k - a_k \cdot \hat{\mathbf{g}}_k(\mathbf{P}_k) \tag{2.2}$$

In Equation 2.2 one can find the particularities of the method that distinguish it from the classic finite differences gradient:

- The estimated gradient $\hat{\mathbf{g}}(\mathbf{P}_k)$ is calculated as shown below:

$$\hat{\mathbf{g}}(\mathbf{P}_k) = \frac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k - c_k\boldsymbol{\Delta}_k)}{2c_k} \cdot \begin{pmatrix} \Delta_1^{-1} \\ \vdots \\ \Delta_N^{-1} \end{pmatrix}_k = \begin{pmatrix} \frac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k - c_k\boldsymbol{\Delta}_k)}{2c_k\Delta_1^k} \\ \vdots \\ \frac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k - c_k\boldsymbol{\Delta}_k)}{2c_k\Delta_N^k} \end{pmatrix}$$

  where $\boldsymbol{\Delta}_k$ is a random perturbation N−dimensional vector with $\Delta_i, \forall i$ is independent identically distributed random variable satisfying $\mathbb{E}(\Delta_i) = 0$ and $\left|\mathbb{E}\left(\left(\Delta_i^{-1}\right)^n\right)\right| < \infty, \forall n$. Note that only two evaluations are required.

- On the other hand, the spacing coefficient c_k is a decreasing sequence of positive real values.

- The step size a_k is also a decreasing sequence of positive real values.

In order to ensure the almost sure convergence of the method to the minimum of the function f_E, some regularity conditions for f_E, $\mathbf{P}_k$, c_k, a_k, $\forall k$ have to be accomplished.

Spall also recommends particular sequences, a_k, c_k, and perturbations, $\boldsymbol{\Delta}_k$, which ensure convergence to the minimum and are used in many similar problems such as [7, 19]. These sequences and perturbations are the following:

$$\begin{cases} a_k = & \frac{a}{(A+k+1)^\alpha} \\ c_k = & \frac{c}{(k+1)^\gamma} \\ \Delta_i \sim & Be(1/2, \pm 1) \end{cases} \tag{2.3}$$

where parameters a, A and c are fixed and depend on the problem, and $\alpha = 0.602$ and $\gamma = 0.101$ because they are the lowest values that satisfy the theoretical conditions as proved in [14]. $Be(1/2, \pm 1)$ is a Bernoulli distribution with probability of $1/2$ for each ± 1 outcome. Note that a_k and c_k are positive sequences that satisfy that

$$\lim_{k\to\infty} a_k = 0\,,\ \lim_{k\to\infty} c_k = 0\,,\ \sum_{k=0}^{\infty} a_k = \infty\,,\ \sum_{k=0}^{\infty} \left(\frac{a_k}{c_k}\right)^2 < \infty$$

which is one of the regularity conditions that ensure the convergence of the method. Further details and the completed proof can be found in [14].

The iterative method runs since some stopping criteria or a finite number of iterations is reached. Therefore, the algorithm can be summarized in Algorithm 1.

1 Remark Many numerical approximations can be done in finite differences gradient computation. For example, one can compute the gradient of a function with an asymmetric gradient such as

$$\nabla f(x) \approx \frac{f(x+h) - f(x)}{h}$$

The same approximation is valid in SPSA algorithm and it will be used in this work:

$$\hat{\mathbf{g}}(\mathbf{P}_k) = \frac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k)}{c_k} \cdot \begin{pmatrix} \Delta_1^{-1} \\ \vdots \\ \Delta_N^{-1} \end{pmatrix}_k = \begin{pmatrix} \dfrac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k)}{c_k\Delta_1^k} \\ \vdots \\ \dfrac{f_E(\mathbf{P}_k + c_k\boldsymbol{\Delta}_k) - f_E(\mathbf{P}_k)}{c_k\Delta_N^k} \end{pmatrix}$$

Algorithm 1 SPSA

% $f_E(\cdot)$ is the objective function for a simulation with certain parameters.
% $\mathbf{P}_0$ is the N-dimensional vector of initial values for the parameters.
% STOP is a True-False value for stopping the algorithm.
% M is the maximum number of iterations allowed.
% $\alpha = 0.602$ and $\gamma = 0.101$

% Set SPSA coefficients
set a, A, c

% Set counter to 0
set $k \leftarrow 0$
set $\mathbf{P}_1 \leftarrow \mathbf{P}_0$

while not STOP **do**
  $k \leftarrow k + 1$
  $c_k \leftarrow c/(k+1)^{\gamma}$
  $a_k \leftarrow a/(A+k+1)^{\alpha}$
  generate $\boldsymbol{\Delta}_k \sim Be(1/2, \pm 1, N)$

  $\mathbf{P}_k^+ \leftarrow \mathbf{P}_k + c_k \boldsymbol{\Delta}_k$

  evaluate $f_k^+ \leftarrow f_E\left(\mathbf{P}_k^+\right)$
  evaluate $f_k \leftarrow f_E\left(\mathbf{P}_k\right)$

  $\hat{\mathbf{g}}_k \leftarrow \left[(f_k^+ - f_k)/c_k\right]/\boldsymbol{\Delta}_k$
  $\mathbf{P}_{k+1} \leftarrow \mathbf{P}_k - a_k \cdot \hat{\mathbf{g}}_k$

  if stopping criteria **then**
    STOP $\leftarrow$ True
  end if

  if $k > M$ **then**
    STOP $\leftarrow$ True
  end if

end while

3
Models for Traffic Simulation

This chapter reviews the theory of the Models for Traffic simulation. It stresses the importance of having traffic models and shows the *core models* used in Traffic Modeling. Finally, the Calibration and Validation step adapted to traffic simulation models is described.

3.1 Traffic Models

Traffic flow modeling through time is object of study for many governments, city councils and research centers since the building of new roads and actions and interventions in a real network are usually expensive investments and affect many people. Having a valid model for a traffic network allows to experiment different possible new scenarios, to forecast the effect of some closures or constructions and are decisive in important decisions making.

3.1.1 Some knowledge about Traffic Flow Theory

A Traffic network is a system compounded principally by some vehicles with a trajectory in a network of different roads. Traffic Flow Theory give some mathematical relations between three important measures in traffic: flow, density and speed. These mathematical equations give an idealized behavior of traffic flow and states fundamental relationships. A look to [12] is highly recommended for a further details of the basics of Traffic Flow Theory.

Traffic Flow theory takes into account simultaneously two types of variables, those related to each vehicle and those related to the traffic flow. The ones related to the vehicles are the classical informational variables such as the *position*, denoted by x_i, the *speed*, denoted by $v_i = dx_i/dt$ and *acceleration*, denoted by $a_i = d^2x_i/dt^2$, so as to its *length*, l_i. Moreover, some behavioral variables are also taken into account such as the driver's *reaction time* and the *maximum desired speed*.

Relations between vehicles that interact in the road can be done in space and in time by using created variables such as the *space headway*, h_{s_i}, to its predecessor and represented in Figure 3.1:

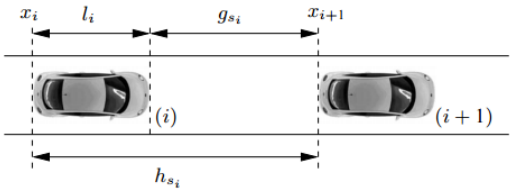


Figure 3.1: Vehicles spcae relation schema. Taken from [12].

In Figure 3.1, two consecutive vehicles are in the same lane in a traffic stream. The follower has a certain space headway h_{s_i} to its leader, equal to the sum of the vehicle's space gap g_{s_i} and its length l_i. Similar relation can be done in time, where the *time headway* is the time between two immediately vehicles' bumpers.

When considering the traffic as a flow, the main variables are the *density*, it measures how crowded the road is, *flow*, it measures the number of vehicles circulating per hour, *occupancy*, it is a time when the measurement location was occupied proportion, *mean speed*, it is the average of all the crossing vehicles' speed in a certain point, and *mean travel times*, which is the mean of the time the vehicles spent to go from a certain point in the road to another.

These last defined variables are the ones easier to capture in a real traffic network, since they can be detected by Radar and Bluetooth sensors.

3.2 Microscopic Modeling of Traffic Flows

As in other fields, traffic networks can be modeled from different levels of detail:

- *Macroscopically*: From an aggregated point of view where only flows of vehicles are characterized as in fluid motion. Thus, one can obtain aggregated macroscopic variables such as density, volume and speed of the flow.
- *Microscopically*: Where the behavior of each vehicle is modeled. In this case, there are many factors to be considered, for example the interactions between the vehicles have to be considered since they affect each others trajectory.
- *Mesoscopically*: It is between microscopic and macroscopic level and tries to combine the advantages of both levels.

Only microscopic modeling will be explained in detail because it is the one that will be used in this work.

As already mentioned, a microscopic point of view in traffic flow modeling is based on the motion of each individual vehicle of the system. There are three main actions in driving that have to be modeled: acceleration, deceleration (or braking) and lane changes. The first models of microscopic traffic flow theory ages from 1950s and all different models try to capture the response of a driver as a function of the time after a lag of time (T) after a stimulus, at time t. That can be represented as:

$$\text{Response}(t + T) = \text{Sensitivity} \cdot \text{Stimulus}(t)$$

Another important part of the Traffic models is the Network elements and structure, which has to be modeled in the computer model since the vehicles have to drive inside the network using selected paths. In this part, one has to take account of the geometry of the network, the traffic control systems and many other aspects such as pedestrians mobility and traffic demands.

3.2.1 Core Models

Car-Following Model

In these models, the stimulus that produces a response of the vehicle modeled comes from the preceding vehicle. The common nomenclature states that the first car is the *leader*, the one who provokes the stimuli, and the second car is the *follower*. Thus, the response of the follower depends, with a certain lag of time, on the behavior of the leader. These models are called *Car-Following* models and many of them have been developed during these last years. The one used in the software platform implementing the current model in this work is the one presented by *P. G. Gipps* in 1981 in [9] and it is a mixture of empirical findings and behavioral assumptions.

In this model, the vehicle pretends to drive at its target speed, function of its desired speed and the limitations imposed by the section and the geometry of the road and the preceding vehicle. Authors of [4] summarises the model in three conditions in the speed of the follower:

1) Vehicle speed does not exceed its maximum desired speed.

2) Vehicle accelerates rapidly until it approaches the desired speed and then acceleration is reduced to almost zero.

3) Vehicle speed is constrained by the vehicle in front. This means that the follower will adjust its speed in order to keep a safe distance between him and the leader.

In Figure 3.2 is shown a diagram of the braking of two vehicles modeled in Car-Following. The leader (vehicle n) starts to brake at time t and it finally stops after a while, at time τ. The follower (vehicle $n + 1$) reacts to the braking with a reaction time lag, T, and starts to brake in order to stop the vehicle leaving a safe distance, L_n.

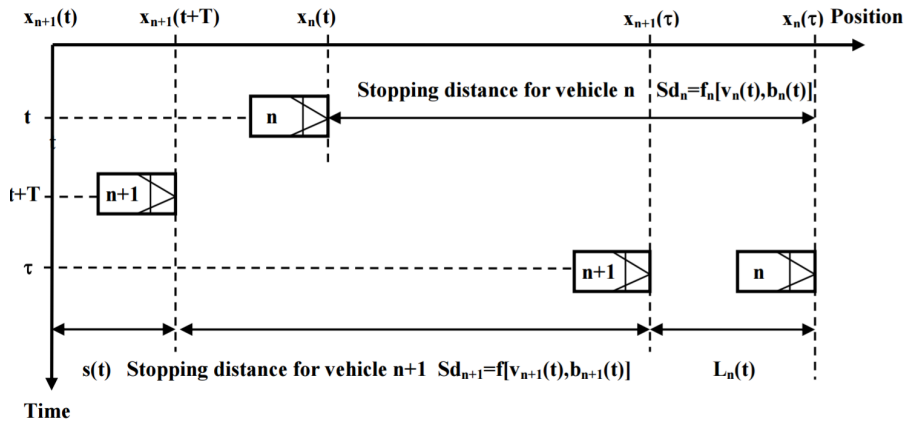


Figure 3.2: Safe deceleration to stop diagram. Taken from [5].

The formal equation for the *follower* speed is the one written above:

$$v_n(t+T) = \min \begin{cases} v_n(t) + 2.5 \cdot a_n \cdot \tau \cdot \left(1 - \frac{v_n(t)}{V_n} \cdot \sqrt{0.025 + \frac{v_n(t)}{V_n}}\right) \\ b_n \cdot \tau + \sqrt{(b_n \cdot \tau)^2 - b_n \left(2 \cdot (x_{n-1}(t) - s_{n-1} - x_{n-1}(t)) - v_n(t) \cdot \tau - \frac{v_{n-1}^2(t)}{\hat{b}}\right)} \end{cases}$$

where a_n, b_n are the maximum acceleration and braking (m/s^2) that the driver wishes to apply, $\hat{b}$ is the estimated maximum braking (m/s^2) of the preceding vehicle $(n-1)$ wishes to apply. $s_{n-1} = L_{n-1}+$ safety gap is the safety distance to the preceding vehicle. V_n is the maximum desired speed of the vehicle and $x_n(t), x_{n-1}(t), v_n(t)$ and $v_{n-1}(t)$ are the current location and speed for both vehicles. Finally, τ is the reaction time, constant for all vehicles. Similar notation applies to Figure 3.2.

Lane-Changing Model

The Lane-Changing model captures the influence of the vehicles in adjacent lanes in the car-following model, [17]. It takes into account the fact that the vehicles in the right-side lane drive slower and this fixes a new desired speed for these vehicles. The model calculates the possible turning options from the current lane and the distance to next turns and compute the improvement chance and the possibility of changing from one lane to another.

This model also takes into account the lane changes, as the name indicates. It counts the mean speed of the adjacent lane vehicles and its number of vehicles in a certain effective distance. With these data, a vehicle in the fast lane will reduce its speed expecting the possibility that a slow vehicle could change from the right-side lane and leaving a safety gap to anticipate these possible manoeuvres, as in Figure 3.3.

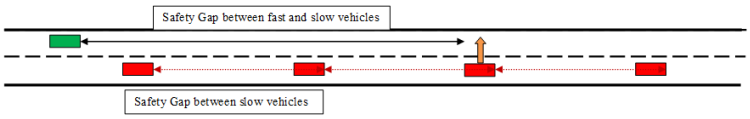


Figure 3.3: Safety Gap in Lane-Changing Model. Taken from [17].

Gap-Acceptance Model

This model determines whether a vehicle can or can not cross when approaching an intersection depending on the nearby vehicles with higher priority, [17]. It also considers the distance between the vehicles to the hypothetical collision point, the speeds of both vehicles and their accelerations and decides if the vehicles can or can not initiate the manoeuvre in the intersection.

Route-Choice Model

The traffic demand in a modeled traffic network is captured by elements called *Origin-Destination (OD) Matrices*. These matrices represent the number of trips between different origins and destinations during a certain time frame. When a vehicle is generated in a microscopic traffic model, an origin and destination are automatically assigned to it regarding the OD matrix.

The Route-Choice model decides the route taking into account the starting time of the trip, the cost of the network sections and the traffic congestion between the fixed origin and destination and calculates the time-frame route that is assigned to the vehicle.

3.3 Calibration & Validation in Traffic Simulation Models

Calibration & Validation is an important step to finally generate a valid model. In the special case of the traffic models for simulation, one has to be able to properly model the input data and to generate the properly simulated outputs to compare with the real data. These two assumptions are really requirements in Traffic Modeling Calibration and Validation. Furthermore, microscopic traffic models combines a lot of uncertainty with lots of parameters, which gives a high importance to this step.

In traffic systems, one can obtain directly observed data such as measurements of flow, speed, travel times, etc. from appropriate devices. These data has to be checked and correctly treated before using them to prepare input data for the computer model or to generate not-directly observable data such as the OD matrices, that can be derived by treating obtained flow data from radar points and travel times between pairs of Bluetooth sensors, for example.

Traffic models are complex and have a large number of different parameters that affect to the performance of the model. Calibration and Validation can be a nested and disgusting process if one decides to do it entirely in one step. Researchers

in Traffic Models for Simulation highly recommend to decompose the Calibration procedure in stages:

1) *Error Checking*: The available data from the transportation network have to be reviewed for errors. This step is necessary to be done before proceeding with calibration. It is aimed at adjusting the default driving behavior parameters for typical road sections.

2) *Capacity Calibration*: One has to perform an initial calibration to identify the values to best reproduce observed traffic capacities in the system. It specializes in fine tuning site specific driving behavior parameters at critical locations.

3) *Route Choice Calibration*: Route choice is important in networks where alternative routes are available. In this case a second calibration process is performed, but this time with the route choice parameters.

4) *Performance Validation*: Finally, the overall model is calibrated and validated by using the measurements such as flow, speed and travel times.

These are general steps of calibration that may not apply depending on the characteristics of the system. For instance, a highway section with only one origin and one destination does not require a *Route Choice* calibration since there is only one possible route.

4
Calibration of Models for Traffic Simulation

In this Chapter, model parameters involved in Car-Following and Lane-Changing modeling in microscopic traffic simulation will be described. These parameters are the optimization variables in the SPSA calibration procedure developed in the work. Firstly, the parameters involved in the Calibration will be explained. Secondly, the Objective Function to minimize and, finally, the Trust Region where the parameters have to remain to have real sense.

4.1 Parameters involved

AIMSUN, one of the most used software products in traffic simulation and the one used in this thesis, considers core models for drivers' behavior: Car-following, Lane-Changing, Gap-Acceptance and Route Choice models. It assigns values to the behavioral parameters using, in most of them, truncated Normal distributions such as the one plotted in Figure 4.1. The software provides default values of mean, standard deviation, minimum and maximum for these parameters, but they can be all modified.

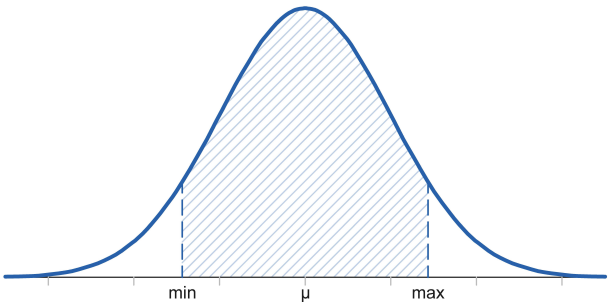


Figure 4.1: Truncated Normal Distribution

After an analysis of the parameters that affect to the core models, the mean and standard deviation of the behavioral parameters that are going to be calibrated with the SPSA method are 12, which are listed below:

1) **Maximum Desired Speed** (Mean): It is the maximum speed that applies to a vehicle class modeling a driver type when no speed restriction is active in sections along the path, either because speed limit is over its desired speed or no congestion effects are present. It is measured in km/h. In other words, a vehicle driving freely without any speed limit on the road and any vehicle affecting its behavior would try to drive at its maximum desired speed. For instance ,in a normal road with 90 km/h speed limit and with maximum desired speed at 110 km/h which means that if this limit exists this vehicles would drive at 90 km/h. If there was no limitation, then it would be driving at 110 km/h.

2) **Maximum Desired Speed** (Standard Deviation): The Standard Deviation of the Maximum Desired Speed distribution. Measured in km/h.

3) **Speed Acceptance** (Mean): It quantifies how much the driver accept the speed limit of the road. It is a non-negative parameter around 1 with the following meaning: Below 1 means that she or he will follow the limits, while above 1 means that the driver will not respect the limit. It is a dimensionless measure.

4) **Speed Acceptance** (Standard Deviation): The Standard Deviation of the Speed Acceptance distribution.

5) **Clearance** (Mean): It is the distance that a vehicle keeps between itself and the preceding vehicle when stopped. It is measured in meters.

6) **Clearance** (Standard Deviation): The Standard Deviation of the Clearance distribution. Measured in meters.

7) **Reaction Time**: It is the time it takes a driver to react to speed changes of the preceding vehicle. Measured in seconds.

8) **Reaction Time at Stop**: It is the time it takes for a stopped vehicle to react to the acceleration of the vehicle in front.

9) **Margin for Overtaking Manoeuvre** (Mean): It is the safety time gap between the overtaking car and the oncoming car. Measured in seconds.

10) **Margin for Overtaking Manoeuvre** (Standard Deviation): The Standard Deviation of the Margin for Overtaking Manoeuvre distribution. Measured in seconds.

11) **Gap** (Mean): It is the distance the follower leaves between him and the leader. Measured in meters.

12) **Gap** (Standard Deviation): The Standard Deviation of the Gap distribution. Measured in meters.

For the remainder of this work, the nomenclature $\mathbf{P} = (p_1, \cdots, p_{12})$ will be used to call the 12 parameters, according to the nomenclature used in Chapter 2.

As already mentioned, there are many parameters in a microscopic simulation model. All other parameters and the minimum and maximum of the truncated distributions have been fixed at the Software default value. These parameters have been chosen taking into account the characteristics of the system studied, described in Chapter 5.

4.2 Objective Function

Three important traffic measures are listed below:

- Flow: (veh/h) The number of vehicles crossing the Radar sensor per hour.
- Speed: (km/h) The average speed of the vehicles crossing the Radar sensor.
- Travel Time: (seconds) The average time needed for going from one Bluetooth point to another of a sample of vehicles.

The error function, f_E following the nomenclature of Chapter 2, selected is a combination of three errors function, one for each measure, flow, speed and travel times. Therefore,

$$\begin{aligned} f_E(\mathcal{R},\mathcal{S}) = f_q(q_{1,\tau_1},\ldots,q_{n_R,\tau_T};\, \hat{q}_{1,\tau_1},\ldots,\hat{q}_{n_R,\tau_T}) + \\ + f_v(v_{1,\tau_1},\ldots,v_{n_R,\tau_T};\, \hat{v}_{1,\tau_1},\ldots,\hat{v}_{n_R,\tau_T}) + \\ + f_t(t_{1,\tau_1},\ldots,t_{n_B,\tau_T};\, \hat{t}_{1,\tau_T},\ldots,\hat{t}_{n_B,\tau_T}) \end{aligned}$$

where q_{i,τ_j}, v_{i,τ_j}, $\forall i = 1,\ldots,n_R$, $j = 1,\ldots,T$ stand for real flows and real speeds for each of n_R Radars at each of the T intervals. Analogously, t_{i,τ_j}, $\forall i = 1,\ldots,n_B$, $j = 1,\ldots,T$ stand for real travel times for the n_B different pairs of Bluetooth at each of the T intervals. The same notation with the hat indicates the respective simulated values.

There are many discrepancies error measures that can be used to show the differences between real and simulated data. The one selected in this thesis for the three measures is the *Normalized Root Mean Square Error* (NRSME), which can be written as follows:

$$NRMSE(\mathbf{x},\hat{\mathbf{x}}) = \frac{\sqrt{\dfrac{\sum_{i=1}^{n}(x_i-\hat{x}_i)^2}{n}}}{\max(\mathbf{x})-\min(\mathbf{x})}\cdot 100 \tag{4.1}$$

where $\mathbf{x}$ is a vector with the observed data and $\hat{\mathbf{x}}$, the correspondent simulated data.

The NRSME was chosen because it represents the standard deviation of the differences between reality and simulation. It is very important to normalize the RMSE because the three measures have different units and magnitudes and, normalizing, they are weighted all them with the same importance.

Therefore, the objective value is finally:

$$f_E(\mathcal{R},\mathcal{S}) = NRMSE(\mathbf{q},\hat{\mathbf{q}}) + NRMSE(\mathbf{v},\hat{\mathbf{v}}) + NRMSE(\mathbf{t},\hat{\mathbf{t}})$$

4.3 Trust Region

All parameters described before have a window of reasonable values, for example, speed, in a highway, has to be between 85 and 120 km/h. The Trust Region of feasible values will be built by the doing the cartesian product of the intervals for each parameter, so:

$$\Omega = [a_1, b_1] \times \cdots \times [a_{12}, b_{12}] \subseteq \mathbb{R}_+^{12}$$

Note that $\Omega \subseteq \mathbb{R}_+^{12}$ because none of the parameters described can be negative values since they are real measures of time and distances. Therefore, $0 < a_i < b_i, \forall i = 1, \ldots, 12$.

In this first approach of SPSA, the manner to incorporate the Trust Region to the SPSA algorithm is to project the vector of parameters $\mathbf{P} = (p_1, \ldots, p_{12})$ orthogonally to the boundary of Ω, as done in [7]. Since Ω is a rectangle in $\mathbb{R}_+^{12}$, the projection only requires to adjust the parameter which is outside its interval to the boundary, therefore:

$$p_i := \begin{cases} a_i & \text{if } p_i \leq a_i \\ p_i & \text{if } a_i < p_i < b_i \\ b_i & \text{if } p_i \geq b_i \end{cases} \quad , \quad \forall i = 1, \ldots, 12$$

5
Site Description

This chapter describes the selected site, defines the location, the available data and the simulation software used to build the model. Firstly, the characteristics of the highway selected will be explained. Secondly, the data collected by sensors during Spring 2015 will be explained and analyzed. The third section of this chapter will be dedicated to explain briefly the model with the improvements done in this project.

5.1 The highway

The road E4 is an European highway that lengths 1590 km from Helsingborg, in Sweden, to Tornio, in northern Finland, crossing vertically all Sweden, [1]. It passes near the capital, Stockholm, and is the most important highway in Sweden connecting important Swedish cities such as Malmö, Linköping, Stockholm, Uppsala.

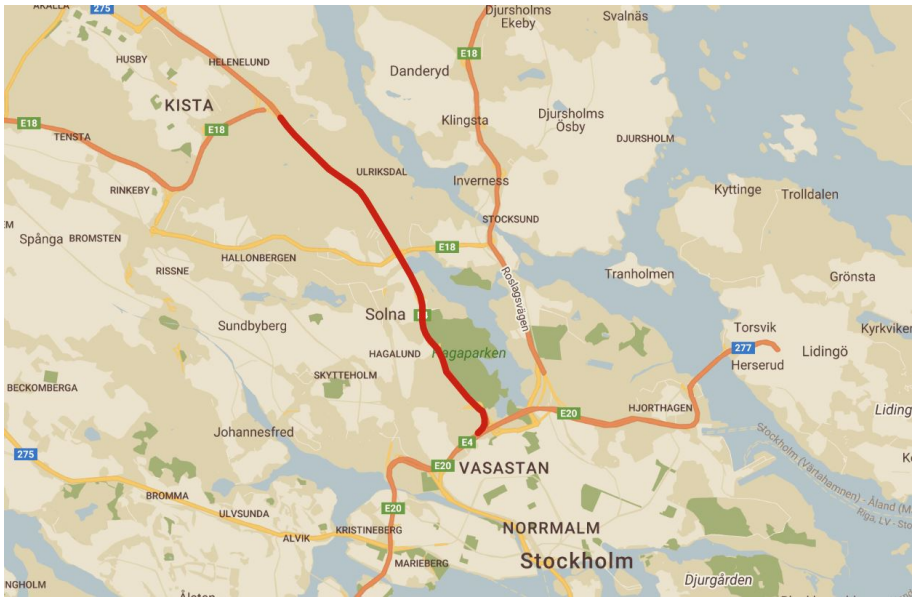


Figure 5.1: The selected section of the Highway near Solna in Stockholm region. Taken from Google Maps.

A section of almost 7.5 km of this highway has been selected in order to build a model. This section connects Stockholm with its most important airport, Arlanda. The selected section near the city of Solna, at the north of Stockholm and shown in Figure 5.1, is intensively used daily by commuters living in the outbacks of Stockholm working downtown.

The election of this site relies on the importance of this section of the highway since it is one of the main entrances to Stockholm and because it is adequately equipped with a sensors network that provide a big amount of data, essential for the calibration of a simulated model.

The section is almost straight throughout the way and it ends entering in a tunnel where it turns significantly to the right. The section has between 3 and 4 lanes and, during these 7.5 km, there are 5 entry lanes and 3 exit lanes where the vehicles can go in or go out the highway, respectively.

5.2 Available data

The available data for this section of the highway were supplied by the network of Bluetooth and Radar sensor depicted in Figure 5.2. These data were collected during March, April and May 2015.

ID	Type	Latitude	Longitude	km point	Distance to 1st	Road
1213	Radar	59.399	17.977	65.420	0.000	Primary
4002	Bluetooth	59.394	17.987	64.970	0.450	Primary
1212	Radar	59.394	17.988	64.970	0.450	Primary
1209	Radar	59.390	17.997	64.265	1.155	Primary
3998	Bluetooth	59.389	18.000	64.090	1.330	Primary
3999	Bluetooth	59.385	18.005	63.580	1.840	Primary
4000	Bluetooth	59.381	18.010	63.040	2.380	Primary
1204	Radar	59.381	18.010	63.025	2.395	Primary
1205	Radar	59.381	18.010	63.025	2.395	Out
1203	Radar	59.379	18.012	62.805	2.615	Primary
4001	Bluetooth	59.373	18.018	62.220	3.200	Primary
1200	Radar	59.372	18.019	61.890	3.530	Primary
4003	Bluetooth	59.366	18.021	61.395	4.025	Primary
1196	Radar	59.367	18.021	61.330	4.090	Primary
1197	Radar	59.367	18.020	61.330	4.090	Out
1194	Radar	59.365	18.023	61.035	4.385	Primary
4004	Bluetooth	59.362	18.026	60.645	4.775	Primary
3241	Bluetooth	59.357	18.032	60.060	5.360	Primary
1187	Radar	59.356	18.034	59.835	5.585	Primary
1178	Radar	59.349	18.035	58.895	6.525	Primary

Table 5.1: Radar and Bluetooth sensors location.

This network consists in 12 Radar and 8 Bluetooth sensors well distributed along the studied section. In Table 5.1, the different locations are summarized, giving

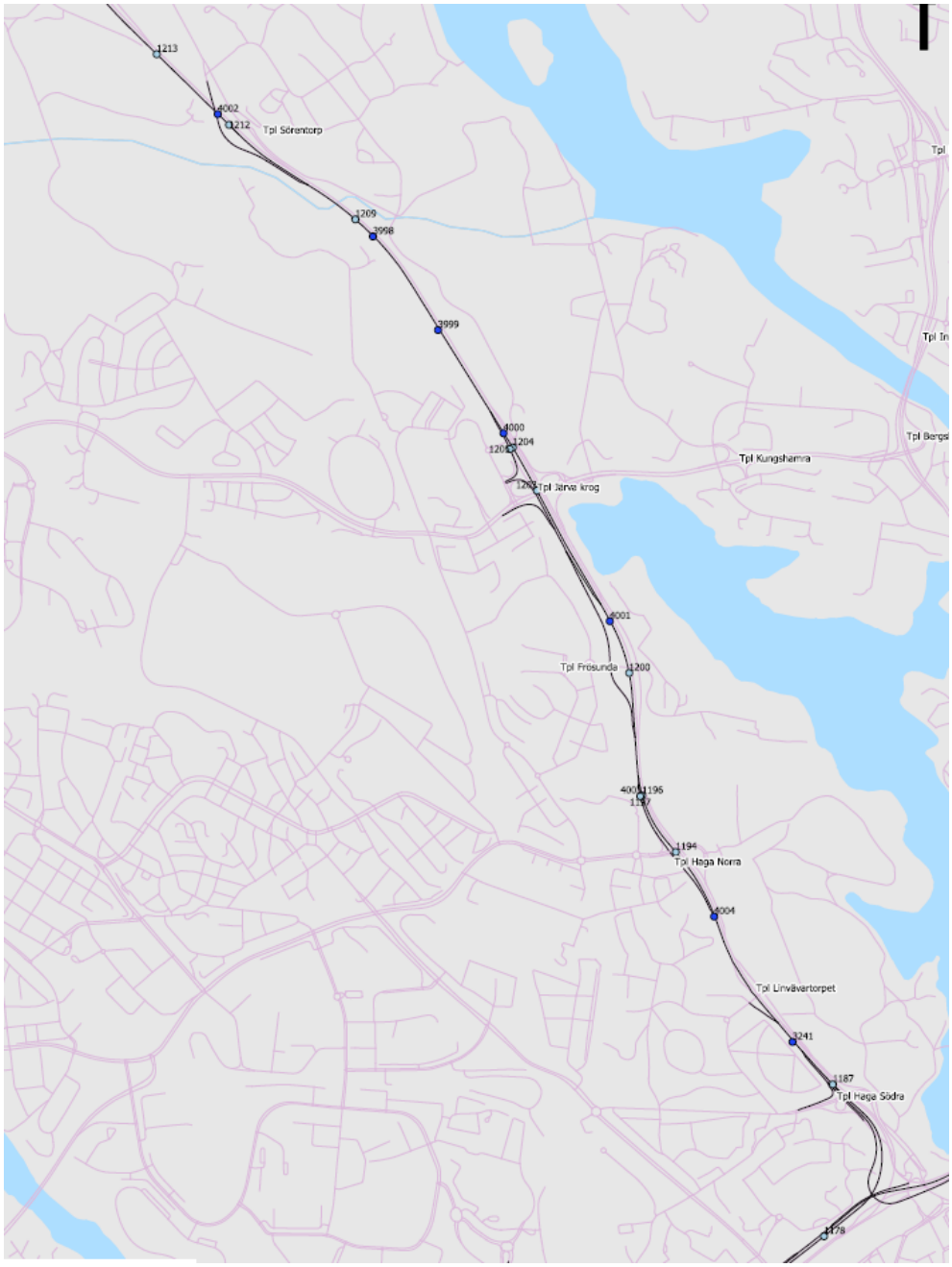


Figure 5.2: Network of sensors. In light blue the Radars are shown, while in dark blue there are the Bluetooth sensors.

information of the location and the distances between them. There are also the kilometer points which permit to know the real distance between sensors, very useful in Bluetooth data as will be explained in Section 5.2.2.

The data collected by the 20 measuring points in the highway was aggregated at minute level. In order to obtain narrow, but computationally less expensive, intervals, aggregations to 5 minutes level have been done.

The two types of data recorded will be explained and analysed.

5.2.1 Radar data

A radar records the flow, in vehicles per hour (veh/h), and the average speed of the vehicles, in kilometer per hour (km/h), at each minute. As mentioned before, there are 12 radar distributed in the section that have been recording the flow and the average speed every minute during three months. In Table 5.2, an extraction of the raw data is shown.

ID	Date	Time	Flow	Speed
1178	2015-03-21	06:00:00	540	76.415
1187	2015-03-21	06:00:00	600	79.361
1194	2015-03-21	06:00:00	360	77.593
1196	2015-03-21	06:00:00	540	81.000
1197	2015-03-21	06:00:00	60	70.000
1200	2015-03-21	06:00:00	600	77.000
⋮	⋮	⋮	⋮	⋮
1204	2015-03-25	20:59:00	1860	82.786
1205	2015-03-25	20:59:00	180	68.000
1209	2015-03-25	20:59:00	1080	89.124
1212	2015-03-25	20:59:00	540	74.713
1213	2015-03-25	20:59:00	840	90.054

Table 5.2: Extraction of the Data collected by the Radars.

In Figure 5.3, the flows of the first radar section are plotted during two different days. As one can easily check, the flow on the working day is always greater than the weekend day. Moreover, the flow between 6AM and 8AM in the working day is very high, which agrees with the fact that people use to go to work to Stockholm at this time.

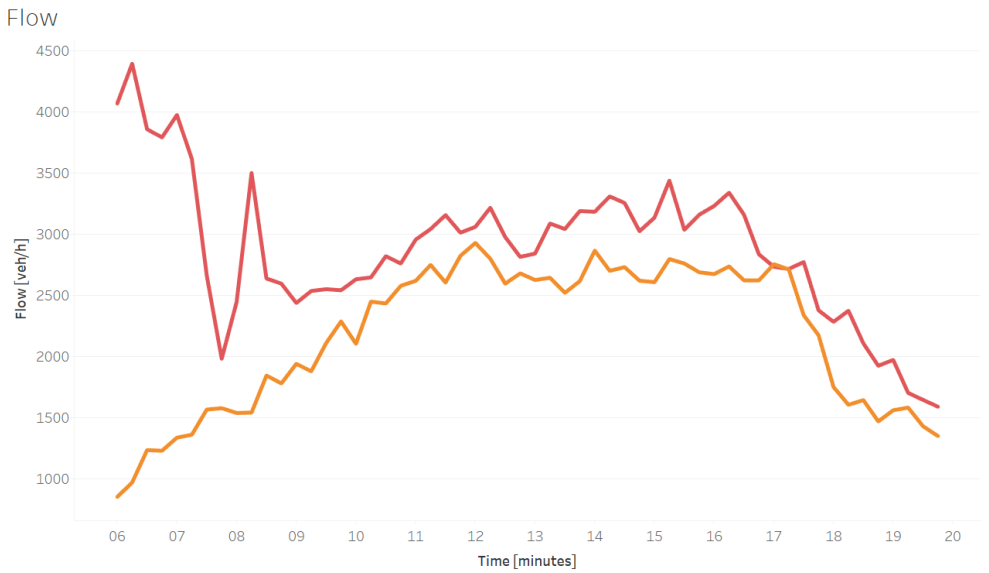


Figure 5.3: Flow at the first sensor in March 21st and 25th.

The average speed along the 7.5 km section is plotted in Figure 5.4. Vehicles arrive to the studied section at a fast speed, more or less 90 km/h, and they decrease the speed during the last 3 kilometers. This is because there is a right turn at the end of the section with a tunnel that impulse the drivers to slow down.

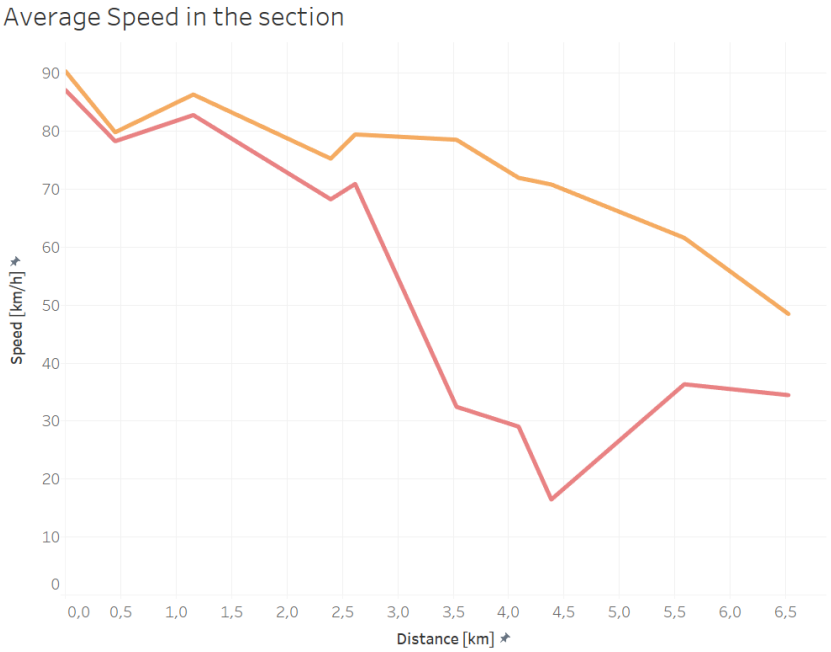


Figure 5.4: Average speed in the section in March 21st and 25th.

5.2.2 Bluetooth Data

The data collected using the Bluetooth points is a bit different. Bluetooth sensors detect certain vehicles and compute the time they spend to go from one measuring point in the highway to another. The distances, in meters, between all Bluetooth sensors along the highway are printed in Table 5.3.

	4002	3998	3999	4000	4001	4003	4004	3241
4002	0	880	1390	1930	2750	3575	4325	4910
3998	880	0	510	1050	1870	2695	3445	4030
3999	1390	510	0	540	1360	2185	2935	3520
4000	1930	1050	540	0	820	1645	2395	2980
4001	2750	1870	1360	820	0	825	1575	2160
4003	3575	2695	2185	1645	825	0	750	1335
4004	4325	3445	2935	2395	1575	750	0	585
3241	4910	4030	3520	2980	2160	1335	585	0

Table 5.3: Distances between Bluetooth points.

The structure of data collected from the Bluetooth is of the form that follows in Table 5.4.

Date	Time	Start Point	End Point	Min	Max	Avg	Median	N
2015-03-21	06:00:00	3998	3999	12	25	19	19	17
2015-03-21	06:00:00	3999	4000	0	0	0	0	0
2015-03-21	06:00:00	4000	4001	0	0	0	0	0
2015-03-21	06:00:00	4001	4003	28	51	36	35	15
2015-03-21	06:00:00	4002	3998	25	41	34	35	17
2015-03-21	06:00:00	4003	4004	36	49	41	40	13
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
2015-03-25	19:59:00	3999	4000	0	0	0	0	0
2015-03-25	19:59:00	4000	4001	0	0	0	0	0
2015-03-25	19:59:00	4001	4003	29	49	38	38	47
2015-03-25	19:59:00	4002	3998	27	55	35	34	49
2015-03-25	19:59:00	4003	4004	34	53	43	42	41
2015-03-25	19:59:00	4004	3241	14	163	25	21	49

Table 5.4: Extraction of the Data collected by the Radars.

In Table 5.4, the numeric columns called *Min*, *Max*, *Avg*, *Median* correspond to times, in seconds, that vehicles last to go from the Starting Point to the Ending Point. On the other hand, N is the sample size.

5.2.3 Consistency of the data

An exercise of validation of the obtained data has been done. As mentioned, the quality of the data is very important. The method to verify the consistency was to

compare two adjacent sensors and understand what is happening between them. An example is shown in Figure 5.5:

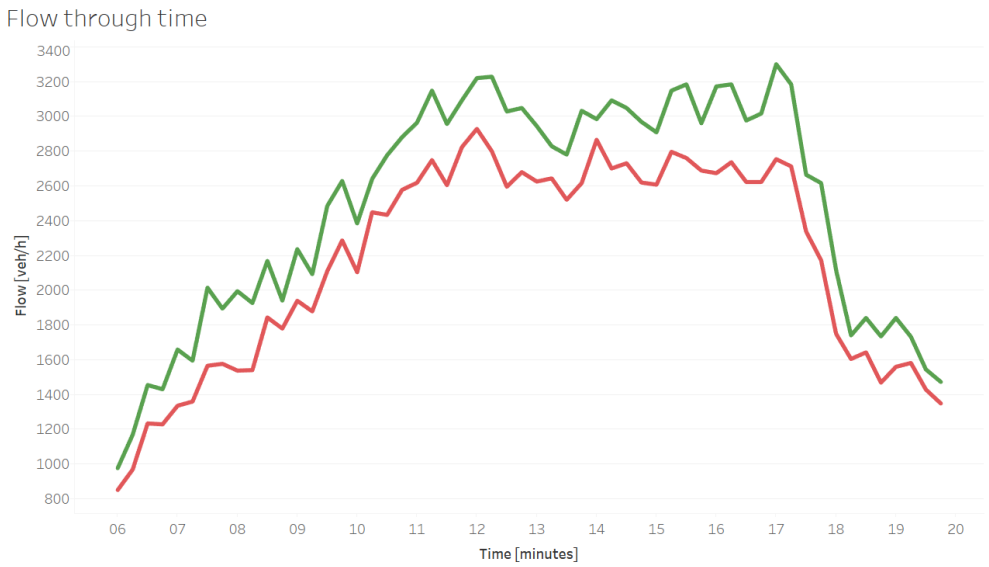


Figure 5.5: Flows through time of 2 Radars in 21st March.

In Figure 5.5, one can see the flows of two adjacent radars, 1209 and 1204, in red and green respectively. There is clearly a decrease of flow which implies that there is an exit between this two sensors, which could be validated in the map of Figure 5.2.

5.3 The Model

5.3.1 Actual

The model was built in June and July 2015 by *Thérese Wilson* and *Rebecca Nilsson* during an Internship in Barcelona, at UPC-Tech. This model was built in *AIMSUN* a very well-known specialized Traffic Simulation software.

This model permits to simulate the traffic in the highway by giving input flows at the entries and turning percentages at intersections as inputs for the model, this is called *Traffic Demand State*. Other important inputs are the values of behavioral parameters of the model. *AIMSUN* has default values for all these parameters that can be adjusted, or not, as one desires.

5.3.2 Analysis

A preliminary analysis of the model showed some differences between real the highway and the model. For instance, in the model there are only 3 entry lanes instead of

the actual 5 according the map, which could be a simplification done by the authors. It has been impossible to verify it before the writing of this memory.

The *Traffic Demand State* was not consistent at all with the available data. Differences in real input and output flows imply a useless model, so it has to be fixed since the best model is desired.

In *AIMSUN*, one can fix the maximum allowed speed of each section. The Swedish legislation sets these limitations to 110 km/h in E4 and E6, the country highest quality highways, [2]. This limitation is reduced near the cities such as in the section studied, where the limitation is reduced to 90 km/h. The model presented some limitations of 70 km/h which is the most restrictive limit, reserved for tunnels.

Finally, some inconsistencies between the model and the data have been detected. Usually, as in this site, traffic data collected by the authorities does not satisfy a minimum of quality, which is essential to build a useful advanced model. In this case, positions of some sensors do not match the information they provide and some assumptions had to be accepted in order to accept the model.

5.3.3 Improvements

The object of this thesis was not to improve the structural model of this site, however, a few physical changes in the site, and inconsistencies in the supplied model requested the corresponding changes, corrections and fine tuning of the available model.

The first correction done was to adapt the input flows and turns of the model to the available data. A Python code reads the data, previously cleaned and prepared, and build automatically the inputs for *AIMSUN* for a determined day and time window.

Secondly, the sensor layout for the network described in Section 5.2 was incorporated to the model, putting the 12 Radar sensors and the 8 Bluetooth points in the model. They are able to measure the same data observed, which will be useful to compare between reality and simulation.

Furthermore, the speed limitations were adapted to the real data observed. Since real data presented a range of speed between 75 and 100km/h, one decided to fix the limits to 90km/h because limitations imposed were too strong and did not reflect the real behavior at the highway.

Despite the main difference between the model and real world was the number of entry lanes, it has not been changed because, as said before, it has been impossible to check if it is a simplification of the model and it has been also impossible to obtain data to define the traffic state when considering the full 5 access lanes.

6 Innovations in the SPSA

This chapter presents the innovations in SPSA algorithm implemented in this thesis. The first section reports on the anomalies found in the way the standard SPSA performs for the simulation-optimization approach used with the simulation model proposed in this thesis. The anomalous behavior is illustrated with the recorded data and their causes are identified, and explained. The second section discusses the proposed innovative changes to the standard SPSA, from which an improved SPSA, better suited to the specific target problem, has been defined. The section reports the new version and corresponding algorithm implementing it.

6.1 Criticism to the first approach of SPSA

The first implementation of the SPSA applied to the model described in Chapter 5, with the aim of optimizing the objective function with respect to the 12 parameters described in Chapter 4, did not do a good performance.

Further analysis of the performance was done once reviewed the codes and the inputs and outputs of the model. Everything seemed to be in order but results remained bad. Also a research of similar problems in similar models was done in order to find solutions to the problem but there were no solutions to the problems presented in this case.

Finally, we conducted an exhaustive analysis of the numerical results to determine the causes and derive the ways of overcoming the identified drawbacks. Two main causes were found:

1) The variables of the minimizing problem are different measures with clearly different magnitudes.

2) The constraints applied to the parameters, the Trust Region, affect badly to the SPSA performance.

Examples with real situations are shown below.

6.1.1 Magnitudes of Parameters

As already presented, parameter magnitudes affect to the SPSA performance. Writing again all the equations that involved in SPSA algorithm, one can notice that size matters:

$$\begin{cases} \mathbf{P}_k^+ & = & \mathbf{P}_k + c_k \boldsymbol{\Delta}_k \\ \hat{\mathbf{g}}(\mathbf{P}_k) & = & \dfrac{f_E(\mathbf{P}_k^+) - f_E(\mathbf{P}_k)}{c_k} \cdot \begin{pmatrix} \Delta_1^{-1} \\ \vdots \\ \Delta_N^{-1} \end{pmatrix}_k = \begin{pmatrix} \dfrac{f_E(\mathbf{P}_k^+) - f_E(\mathbf{P}_k)}{c_k \Delta_1^k} \\ \vdots \\ \dfrac{f_E(\mathbf{P}_k^+) - f_E(\mathbf{P}_k)}{c_k \Delta_N^k} \end{pmatrix} \\ \mathbf{P}_{k+1} & = & \mathbf{P}_k - a_k \cdot \hat{\mathbf{g}}_k(\mathbf{P}_k) \end{cases}$$

Note that both the spacing coefficient a_k and the value of the estimated gradient $|\hat{\mathbf{g}}(\mathbf{P}_k)_i|, \forall i = 1, \ldots, 12$ are the same values for the 12 parameters and they do not have the same effect for different magnitude parameters.

Figure 6.1 depicts graphically the changes for two different parameters in 10 iterations, while the speed varies slightly inside the Trust Region, another variable changes its value escaping twice from the Trust Region.

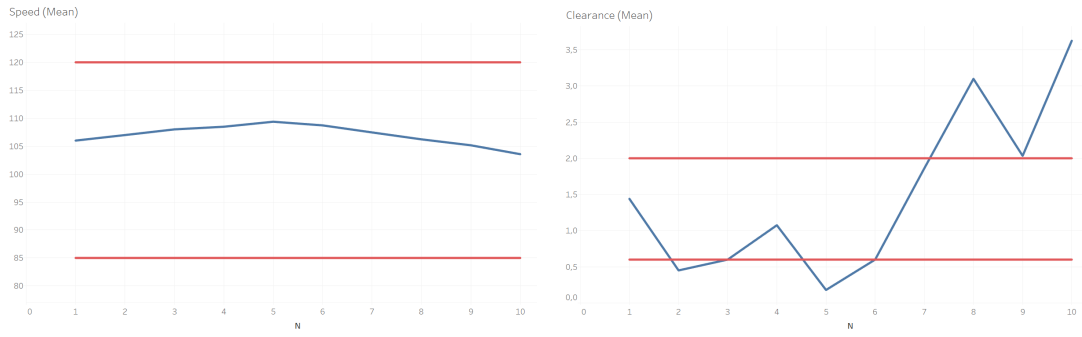


Figure 6.1: Different magnitude parameters evolution.

This clear difference is due to the different magnitudes they have, speed values move around 100 km/h and clearance around 1.5 meters. This fact can lead SPSA to unfeasible values or can affect directly to the convergence of the algorithm if the initial value of higher magnitude parameters is far enough from the minimum associated value. Higher values of a_k, in order to make faster the convergence, are not possible because they could provoke large oscillations of lower magnitude parameters.

6.1.2 Trust Region

The second phenomena was produced by the Trust Region in lower magnitude parameters. As mentioned in Section 4.3, the Trust Region for the parameters was

added to the algorithm checking if the new value remains inside the interval of feasible values and, if it escapes, projecting it to the boundary.

This procedure affects directly to those parameters that vary more than the interval range in first iterations, such as the ones seen in Figure 6.1. In Figure 6.2 it can be shown that, while the speed varies slightly inside the Trust Region, clearance mean swaps its value between a_i and b_i, the limits of the interval in the early iterations.

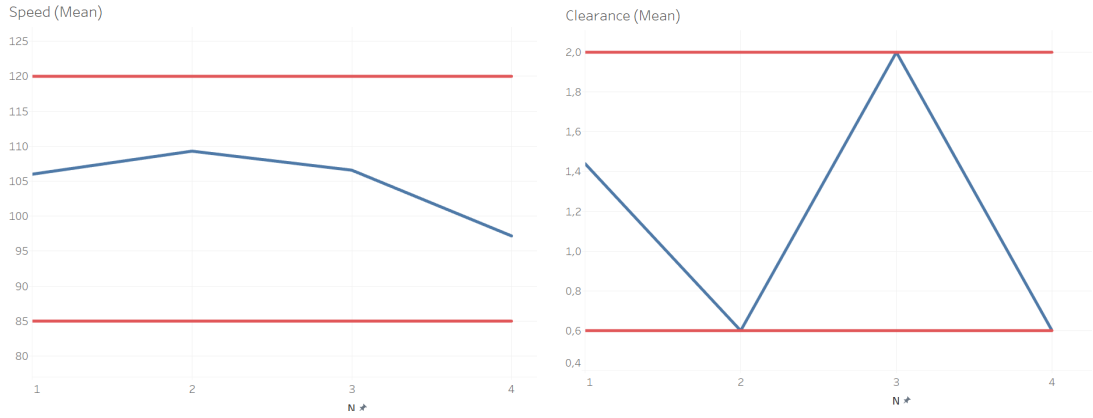


Figure 6.2: Different magnitude parameters evolution with the Trust Region.

As before, this phenomena also contributes to worse the performance of the algorithm and for lower magnitude parameters transform the optimization to a binary optimization of these parameters, at least at the early iterations. Readjusting the SPSA coefficients values provoke later convergences because of the slight changes of the higher magnitude parameters.

2 Remark Results obtained for Figures 6.1 and 6.2 were made by using the same seed and coefficient values than the final algorithm of this work but returning to the original algorithm. Trust Region was eliminated in Figure 6.1, to see clearly the oscillations.

6.2 Innovations in the SPSA

In this section, two solutions for each phenomena are presented, showing the contributions they bring to the algorithm in this model, where the many parameters of different magnitudes are subject to bounding constraints, $p_i \in [a_i, b_i], \forall i = 1, \ldots, 12$, for feasibility and credibility reasons.

These two solutions are associated directly to the two problems described in Section before.

6.2.1 Normalized Parameters

In order to avoid the problem with different magnitudes, they were homogenized to the same magnitude by rescaling them from $[a_i, b_i]$ to $[0, 10]$ with the classical linear interpolation:

$$\begin{array}{rcl} \varphi_i : [a_i, b_i] & \to & [0, 10] \\ p_i & \mapsto & \tilde{p}_i = 10 \cdot \dfrac{p_i - a_i}{b_i - a_i} \end{array} \quad , \ \forall i = 1, \ldots, 12$$

and its inverse transformation:

$$\begin{array}{rcl} \varphi_i^{-1} : [0, 10] & \to & [a_i, b_i] \\ \tilde{p}_i & \mapsto & p_i = a_i + \tilde{p}_i \cdot \dfrac{b_i - a_i}{10} \end{array} \quad , \ \forall i = 1, \ldots, 12$$

Therefore, let $\mathbf{\Phi} = (\varphi_1, \ldots, \varphi_{12})$ and $\mathbf{\Phi}^{-1} = (\varphi_1^{-1}, \ldots, \varphi_{12}^{-1})$ be the correspondent functions from Ω to $\tilde{\Omega} = [0, 10]^{12}$, the minimization problem defined in Equations 2.1 can be written as:

$$\begin{cases} \min & \tilde{f}_E(\tilde{\mathbf{P}}) \\ \text{s. to} & \\ & \tilde{\mathbf{P}} \in \tilde{\Omega} = [0, 10]^{12} \subset \mathbb{R}^{12} \end{cases} \tag{6.1}$$

where $\tilde{f}_E(\tilde{\mathbf{P}}) = f_E \circ \mathbf{\Phi}^{-1}$. Once the minimum has been found, denoted as $\tilde{\mathbf{P}}^*$, one has to undo the normalization, that is $\mathbf{P}^* = \mathbf{\Phi}^{-1}\left(\tilde{\mathbf{P}}^*\right)$.

6.2.2 Penalized Objective Function

The problem with the Trust Region was not solved by using Normalized parameters as variables for the minimization problem. Other alternatives have been contemplated and finally the alternative presented by *I. J. Wang* and *J. C. Spall* in [18] of using penalized objective function was chosen as the best option.

Penalty-function approach alternative consists on transforming the constrained minimizing problem to a free minimizing problem by modifying the objective function, in a more formal way:

$$(P1) = \begin{cases} \min & f(\mathbf{x}) \\ \text{s. to} & \\ & \mathbf{x} \in \Omega \subset \mathbb{R}^N \end{cases} \rightsquigarrow (P1^*) = \begin{cases} \min & f(\mathbf{x}) + r \cdot P(\mathbf{x}) \\ \text{s. to} & \\ & \mathbf{x} \in \mathbb{R}^N \end{cases}$$

where $P(\mathbf{x}) \geq 0$ is a penalty function that penalizes those $\mathbf{x} \notin \Omega$ and r is a positive real number such that the minimum of the new problem $(P1^*)$ is the same that in the original problem $(P1)$.

3 Remark This second improvement was implemented after applying the Normalization of Parameters described before. So the problem is the one described in Equations 6.1 and, thus, notation in this section includes a tilde.

Following the nomenclature of [18], since $\tilde{\Omega} = [0, 10]^{12}$, the constraints are all of the form:

$$0 \leq \tilde{p}_i \leq 10, \ \forall i = 1, \ldots, 12$$

which can be split in 12 pair of constraints:

$$\begin{cases} -\tilde{p}_i \leq 0 \\ \tilde{p}_i - 10 \leq 0 \end{cases} \forall i = 1, \ldots, 12 \tag{6.2}$$

Furthermore, another constraint have been added in order to give more real sense to the results. This constraint relate two variables, *Reaction Time* and *Reaction Time at Stop*. Reaction Time at Stop is usually greater than Reaction Time since, when stopped, one has to engage and the process to move is slower in overall. Therefore, the 25th constraint is:

$$\varphi_7^{-1}(\tilde{p}_7) - \varphi_8^{-1}(\tilde{p}_8) \leq 0 \tag{6.3}$$

And finally, the feasible set, or Trust Region, can be written as follows:

$$\tilde{\Omega} = \left\{ \tilde{\mathbf{P}} \in \mathbb{R}^{12} : q_j(\tilde{\mathbf{P}}) \leq 0, \ \ j = 1, \cdots, 25 \right\}$$

where q_j are the left-hand side of the 25 inequalities written in Equations 6.2 and 6.3.

There are many penalty functions that are used in many problems, as one can check in Section 2.1 of [18]. The selection of the penalty given to the points that do not satisfy the constraints is an important point that can affect directly to the convergence.

The one selected in this thesis is the *standard quadratic penalty function*, which can be shown in Figure 6.3 and analytically written as:

$$P(\tilde{\mathbf{P}}) = \sum_{j=1}^{25} \alpha_j \cdot P_j(\tilde{\mathbf{P}}) = \sum_{j=1}^{25} \alpha_j \cdot \max\left\{0, q_j(\tilde{\mathbf{P}})\right\}^2$$

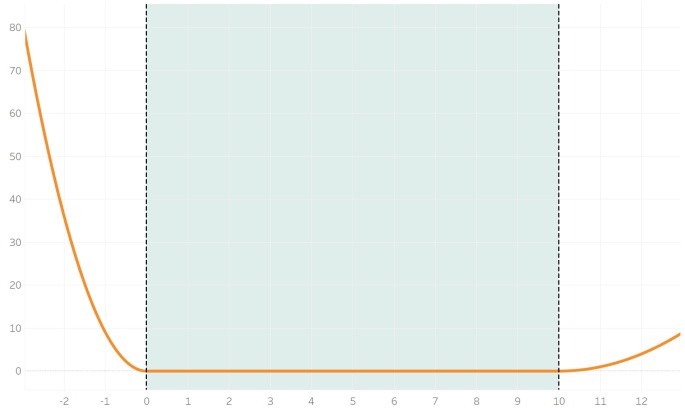


Figure 6.3: The penalty function for one constraint for a normalized parameter ($P_j(\tilde{\mathbf{P}})$).

4 Remark A coefficient α_j was included in the Penalty function in order to give more importance to the lower bound. The reason lies in numerical problems in the

simulator if any of the real behavioral parameters approaches to 0 or negative values. Therefore,

$$\alpha_j = \begin{cases} 1 & \text{if } q_j \text{ is a Upper Bound.} \\ 3 & \text{if } q_j \text{ is a Lower Bound.} \end{cases}$$

For the 25th inequality, the coefficient was set at $\alpha_{25} = 3$.

5 Remark In the final algorithm, a restriction of avoiding non-positive values for all the non-normalized parameters has been incorporated because *AIMSUN* automatically crashes if there is a negative parameter.

At this moment the objective function has been changed and that affects, of course, to the SPSA algorithm because the objective function did change and, therefore, its gradient too. Therefore:

$$\nabla \tilde{f_E}(\tilde{\mathbf{P}}) + r \cdot \nabla P(\tilde{\mathbf{P}}) = \nabla \tilde{f_E}(\tilde{\mathbf{P}}) + r \sum_{j=1}^{25} \alpha_j \cdot \nabla P_j(\tilde{\mathbf{P}}) = \nabla \tilde{f_E}(\tilde{\mathbf{P}}) + r \sum_{j=1}^{25} \alpha_j \cdot \nabla P_j(\tilde{\mathbf{P}}) =$$
$$= \nabla \tilde{f_E}(\tilde{\mathbf{P}}) + r \sum_{j=1}^{25} \alpha_j \cdot \max\left\{0, q_j(\tilde{\mathbf{P}})\right\} \cdot \nabla q_j(\tilde{\mathbf{P}})$$

where $\nabla(\cdot)$ is the gradient with respect to the normalized parameters. As proposed in [18], the actualization of the parameters for the next iteration, Equation 2.2, is modified with the new gradient:

$$\tilde{\mathbf{P}}_{k+1} = \tilde{\mathbf{P}}_k - a_k \cdot \hat{\mathbf{g}}(\tilde{\mathbf{P}}_k) - a_k \cdot r_k \cdot \nabla P(\tilde{\mathbf{P}}_k) =$$
$$= \tilde{\mathbf{P}}_k - a_k \cdot \hat{\mathbf{g}}(\tilde{\mathbf{P}}_k) - a_k \cdot r_k \sum_{j=1}^{25} \alpha_j \cdot \max\left\{0, q_j(\tilde{\mathbf{P}}_k)\right\} \cdot \nabla q_j(\tilde{\mathbf{P}}_k)$$

Since the gradient elements are different to 0 only if the parameter is involved with the constraint, this new added term helps significantly to the parameters that escaped of the Trust Region to come back. It can be seen in a little example:

6.2.1 Example Imagine that in iteration k, only the third parameter $\tilde{p}_3^k$ escapes from the interval $[0, 10]$, let $\tilde{p}_3^k = 11$, $a_k = 1, r_k = 0.8$ and $\tilde{\mathbf{g}}(\tilde{\mathbf{P}}_k) = \pm 0.25$.

In this case, only constraint $\tilde{p}_3^k - 10 = 11 - 10 = 1 \leq 0$ is violated and next step will correct it:

$$p_3^{k+1} = p_3^k - a_k \cdot \tilde{\mathbf{g}}(\tilde{\mathbf{P}}_k)_3 - a_k \cdot r_k \cdot \max\{0, 11 - 10\} =$$
$$= 11 - 1 \cdot 0.25 - 1 \cdot 0.8 \cdot 1 = 9.95$$

Without the contribution of the last term, the normalized parameter would not enter to the Trust Region at the next iteration.

Of course, the mathematical subtraction could not be enough but the new contribution always pushes in the right direction.

6 Remark Note that, another positive real numbers sequence r_k is added in order to ensure convergence, as proposed in [18]. This sequence is taken as $r_k = \dfrac{r}{k^{0.1}}$, where $r > 0$.

6.3 Final Innovated SPSA Algorithm

This section aims to collect the final algorithm with the innovations implemented. It is shown in Algorithm 2 where new contributions are highlighted in purple:

7 Remark Notation used **C25** would represent the following:

$$\mathbf{C25} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{max}(p_7^k - p_8^k, 0) & -\mathbf{max}(p_7^k - p_8^k, 0) & 0 & 0 & 0 & 0 \end{pmatrix}^\top$$

which corresponds to the term associated to the 25th constraint, explained in Equations 6.3, avoiding multiplicative factors which are included, implicitly, in the value of α_{25}

Algorithm 2 Innovated SPSA

% $\tilde{f}_E(\cdot)$ is the penalized objective function for a simulation with certain parameters.
% $\mathbf{P}_0$ is the N-dimensional vector of initial values for the parameters.
% STOP is a True-False value for stopping the algorithm.
% M is the maximum number of iterations allowed.
% Norm($\mathbf{P}$, **TRmin**, **TRmax**) and InvNorm($\mathbf{P}$, **TRmin**, **TRmax**) are the linear
% interpolation and its inverse functions.
% **max**$(\cdot,\cdot)$ returns the maximum component by component.
% **sum**$(\cdot)$ sums all the components of the vector.
% $\alpha = 0.602$ and $\gamma = 0.101$

% Set SPSA coefficients
set a, A, c
set r

% Set counter to 0
set $k \leftarrow 0$
set $\mathbf{P}_1 \leftarrow \mathbf{P}_0$

while not STOP **do**
  $k \leftarrow k+1$
  $c_k \leftarrow c/(k+1)^{\gamma}$
  $a_k \leftarrow a/(A+k+1)^{\alpha}$
  $r_k \leftarrow r/(k)^{0.1}$
  generate $\boldsymbol{\Delta}_k \sim Be(1/2, \pm 1, N)$

  $\tilde{\mathbf{P}}_k \leftarrow \mathtt{Norm}(\mathbf{P}_k, TRmin, TRmax)$
  $\tilde{\mathbf{P}}_k^+ \leftarrow \tilde{\mathbf{P}}_k + c_k\boldsymbol{\Delta}_k$
  $\mathbf{P}_k \leftarrow \mathtt{InvNorm}(\tilde{\mathbf{P}}_k^+, TRmin, TRmax)$

  evaluate $f_k^+ \leftarrow \tilde{f}_E\left(\mathbf{P}_k^+\right)$
  evaluate $f_k \leftarrow \tilde{f}_E\left(\mathbf{P}_k\right)$

  $\hat{\mathbf{g}}_k \leftarrow \left[(f_k^+ - f_k)/c_k\right]/\boldsymbol{\Delta}_k$

  $\tilde{\mathbf{P}}_{k+1} \leftarrow \tilde{\mathbf{P}}_k - a_k \cdot \hat{\mathbf{g}}_k - a_k \cdot r_k \left[\mathbf{sum}(\mathbf{max}(\tilde{\mathbf{P}}_k - 10)) - 3 \cdot \mathbf{sum}(\mathbf{max}(-\tilde{\mathbf{P}}_k)) + 3 \cdot \mathbf{C25}\right]$

  if stopping criteria **then**
    STOP $\leftarrow$ True
  end if

  if $k > M$ **then**
    STOP $\leftarrow$ True
  end if

end while

7
Experimental Design

This chapter explains the different Software used, the initial values and Trust Region selected and the methodology followed in this thesis to obtain the results described in Chapter 8.

7.1 Initial Values and Trust Region

The minimum and maximum values for the parameters were determined from previous knowledge and experience on the Gipps' Model. *AIMSUN* has some default values based on lots of previous experiments that are a good reference and they give a very good idea of the magnitude of these parameters.

	Initial Value	Minimum	Maximum
Maximum Desired Speed Mean	106.00	85.00	120.00
Maximum Desired Speed Std. Dev.	6.80	2.00	10.00
Speed Acceptance Mean	1.12	0.70	1.40
Speed Acceptance Std. Dev.	0.68	0.20	1.00
Clearance Mean	1.44	0.60	2.00
Clearance Std. Dev.	0.72	0.30	1.00
Reaction Time	1.28	0.50	1.80
Reaction Time at Stop	1.60	0.70	2.20
Margin for Overt. Mean	6.80	2.00	10.00
Margin for Overt. Std. Dev.	2.80	1.00	4.00
Gap Mean	0.98	0.20	1.50
Gap Std. Dev.	0.56	0.20	0.80

Table 7.1: Initial values and limits for the 12 parameters.

For the initial value, a non-centered point inside the region has been selected. After some trials, the vector of initial parameters is the one shown in Table 7.1, which corresponds to

$$p_i = \frac{2 \cdot a_i + 3 \cdot b_i}{5}, \quad \forall i = 1, \ldots, 12$$

As stated in Section 6.2, the normalization of the parameters has been done in order to homogenize the same step effect to all the parameters.

7.2 SPSA coefficients

J. Spall describes in [15] a methodology to select correctly the coefficients for the SPSA algorithm. These coefficients have to be very well adjusted because they affect directly to convergence speed and to initial stability of the method.

As recommended, used equations for c_k and a_k are those written before in Equation 2.3:

$$a_k = \frac{a}{(A+k+1)^\alpha}\ ,\ \ c_k = \frac{c}{(k+1)^\gamma}$$

Therefore, only c, a and A have to be fixed, since $\alpha = 0.602$ and $\gamma = 0.101$ are the best values as *Spall* shows in [14].

Recalling recommendations given in [15], it is effective to set c as a small positive number, such as it is in the formula of finite differences. The values of a and A can be chosen together to ensure effective performance of the SPSA algorithm.

One has to choose the largest possible a so as to have significant steps, but the larger it is, slower is the convergence. That's the reason of constant A, which permits larger values of a making more stable the method. The election for these parameters is, usually, done by trial and error and checking in literature for the values of similar experiments.

The new coefficient added in this work, which is defined in Section 6.2, is the initial value for the penalizing sequence in the step size, r. This value has to be set according to the magnitude of the penalization on the objective value.

The used values are finally summarised in Table 7.2:

	Value
c	0.6
a	2.2
A	6.0
r	1.0

Table 7.2: Initial values for SPSA coefficients.

7.3 Real Data Preparation

As mentioned in Section 4.2, the objective function uses three traffic measures: flow, speed and travel times. The network provides these three measures because it combines Radars and Bluetooth points. The temporal horizon for the experiments is a trade-off between computational cost, a lot of simulations have to executed to optimize the objective function, but it has to be long enough to account for the effect

of the selected parameters. A one hour interval is set to achieve both objectives. The temporal horizon has to split into subintervals to account for time-dependent variability of traffic measures and from 1 to 10 minutes are usually considered, thus a 5 minutes subintervals seems to be a reasonable selection.

Real observations cover from 1st March to 31st May, which is a big amount of data. For computational issues, once aggregated to 5 minutes level, two different days and times have been selected. The first day, 19th March 2015 from 10:30AM to 11:30AM, will be used for calibration and the second, 26th March 2015 from 10:30AM to 11:30AM will be used for validation. These two days were both consecutive Thursdays and it has been checked there were no incidences nor congestion during these intervals.

Therefore, there are 12 radars providing 12 measures of flow and speed, 1 hour in 5 minutes intervals equals to 12 measures. This equals to 144 values of flow and 144 values of speed that have to be collected from the simulation. Travel times data is provided by 5 pairs of Bluetooth points, which equals to 60 measures.

All these data have been cleaned and prepared in CSV files to be entered to *MATLAB*, using `RStudio` and *Tableau* to visualize them.

7.4 Obtaining simulated data

AIMSUN allows to place sensors and to equip some vehicles with detectors in order to be detected by the sensors. In the project, the 30% of the vehicles were equipped. This value is considered to be very realistic for the real scenario equipment of in-car navigation devices.

AIMSUN generates a `SQLite` Database with all the simulated data. From two different tables one can obtain the 3 traffic measures doing some queries. *AIMSUN* user's manual, [17], provides all the information about the different tables.

Speed and flow are directly obtained from table `MIDETEC` which contains the flow, speed, occupancy and density for each sensor at each interval of time of the simulation.

For the travel times, the query is a little more sophisticated because table `DETEQUIPVEH` contains, for each sensor and equipped vehicle, when the vehicle cross the sensor, in seconds. Hence, for each pair of Bluetooth sensors of the real network, one has to obtain the times and subtract such as in the query, as example:

```
 select A.oid as start_point, B.oid as end_point, A.idveh as idveh, A.timedet
as time_ini, B.timedet as time_fin, B.timedet - A.timedet as ttime from
(select * from DETEQUIPVEH where oid = 68125 and did = 10) A inner join
(select * from DETEQUIPVEH where oid = 68126 and did = 10) B on A.idveh
= B.idveh');
```

8 Remark Note that in the example, detectors 68125 and 68126 were selected. These are the correspondent ID's for two of the real Bluetooth points in *AIMSUN*. All the correspondences between *AIMSUN* and real ID's can be found in Appendix A.

Since the real data is aggregated at 5 minutes level, the simulated data for travel times obtained with the query has been aggregated by using `time_ini`, the time they cross the first Bluetooth point and obtaining the mean for all the vehicles of the interval.

7.5 Routine

The SPSA routine has been implemented in *MATLAB* due to its versatility. *MATLAB* is powerful working with vectors and matrices, making calls in Python to *AIMSUN* to run two simulations every iteration and also because it can interact with databases easily, as `SQLite`. Each commented agent plays an important role in each iteration.

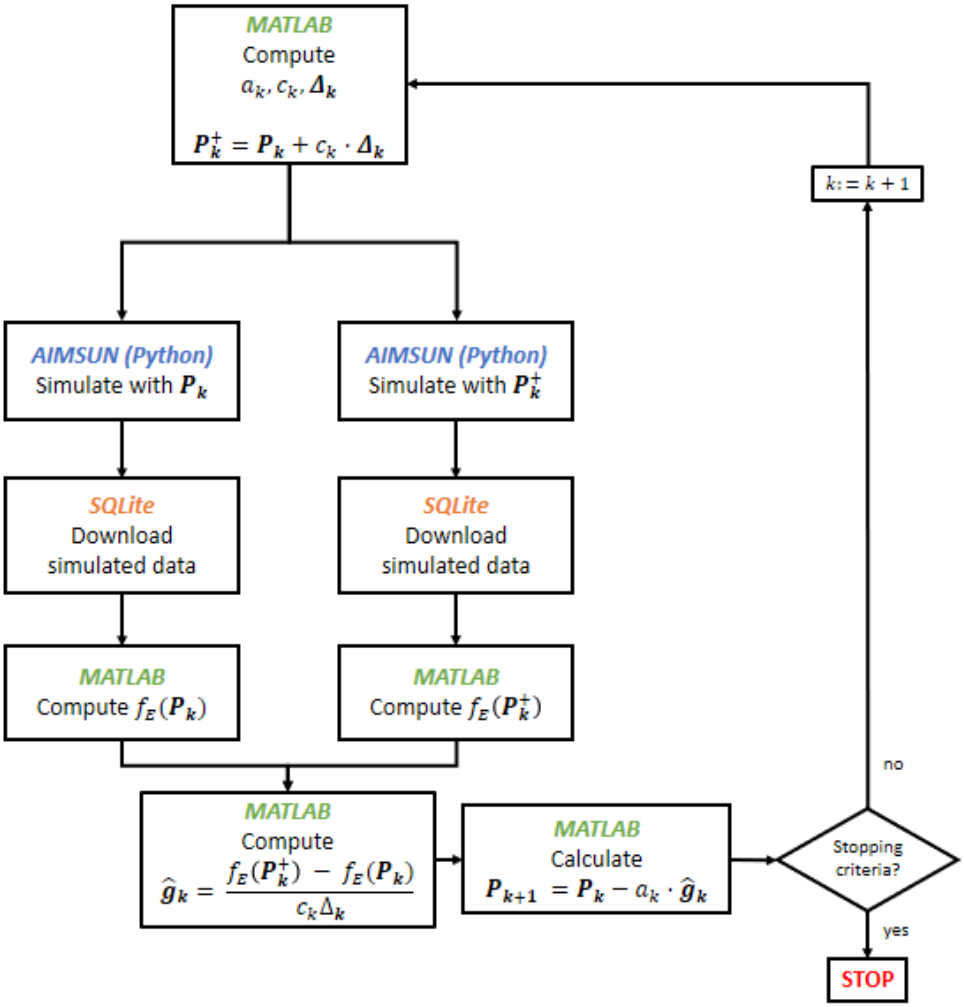


Figure 7.1: SPSA Routine

The full procedure for each iteration can be seen in Figure 7.1. In *MATLAB*, the

particular SPSA coefficients are actualized by using Equations 2.3 and the random perturbations are generated. *MATLAB* runs a Python code which launches a simulation after changing the parameters values to $\mathbf{P}_k$ or $\mathbf{P}_k^+$. These new computed parameters are written to a CSV file and Python reads it to set all the parameters before running the simulation.

Once the simulation is finsihed, *MATLAB* reads the results from `SQLite` and compute the objective function value, comparing them with the corresponding real data obtained from the Network of Radars and Bluetooth points.

Having the objective values for both simulations, the Gradient is directly calculated and next point $\mathbf{P}_{k+1}$ can be calculated. Finally, one stops the algorithm if the maximum number of iterations is reached or if the stopping criteria indicates the convergence of the algorithm.

In this work, the stopping criteria is the following:

$$\frac{\|\mathbf{P}_{k+1} - \mathbf{P}_k\|}{\|\mathbf{P}_{k+1}\|} < e$$

where e is a tolerance value, set at 10^{-6}.

9 Remark All codes can be found in Appendices B, C and D of this thesis.

8
Results Analysis

This chapter explains, shows and analyzes the results of the application of SPSA to the described site. First section will be dedicated to analyze the innovated SPSA performance

8.1 SPSA Performance

Since simulating traffic requires a lot of computational time, the SPSA performance, which launches two complete simulations for each iteration, was computationally expensive in time. It took almost 12 hours to reach a minimum, because there were around 250 iterations and each simulation takes almost 2 minutes. The SPSA algorithm was launched in the model for the 19th of March, the one selected for calibration.

In Figure 8.1 is plotted the objective value evolution during the SPSA performance. The reduction of the objective function is clear, from the maximum value it got, 40.171 at 2nd iteration, to the minimum value, 32.690 at 179th iteration. This equals to a relative drop of 22.88%. Note that the drop is almost linear since it approaches the minimum and then, the function value stabilizes around the minimum.

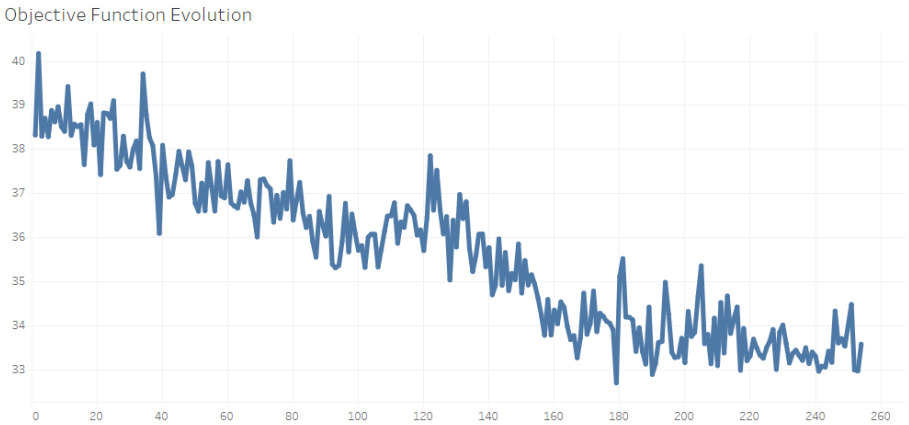


Figure 8.1: Objective Function Evolution in SPSA Performance.

In Figure 8.2, the 12 normalized parameters, $\tilde{p}_i, i = 1, \ldots, 12$ are overplotted with the bounding limits, $[0, 10]$. It can be seen that the first perturbations are larger and parameters separate rapidly from the initial value. Perturbations go smaller at next iterations, since the step a_k is a decreasing sequence. Parameters finally stabilize and oscillate near a value, which is the minimum.

Note that many parameters escape from the Trust Region but the algorithm make them return inside in a low number of iterations. Indeed, there is a parameter which remains a large number of iterations around the boundary, which may indicate that its optimal value is in the boundary.

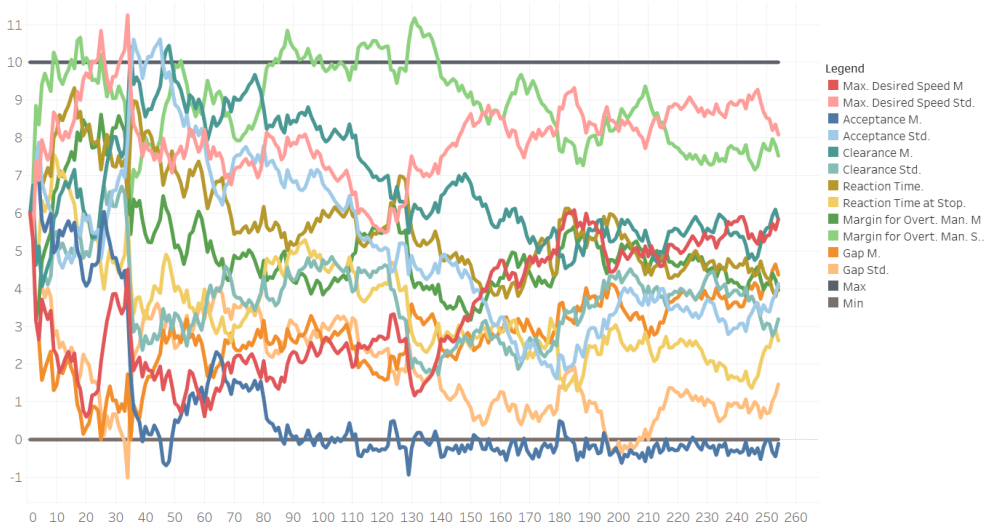


Figure 8.2: Normalized Parameters Evolution in SPSA Performance.

The real value for the parameters which reached the minimum value, at iteration 179 are written in Table 8.1. These are accepted as the optimal values for reaching the minimum objective function. It will be denoted as $\mathbf{P}^* = \mathbf{\Phi}^{-1}(\tilde{\mathbf{P}}^*)$.

	Value	
Maximum Desired Speed (Mean)	102.173	km/h
Maximum Desired Speed (Std. Dev.)	8.540	km/h
Speed Acceptance (Mean)	0.688	
Speed Acceptance (Std. Dev.)	0.329	
Clearance (Mean)	1.368	meters
Clearance (Std. Dev.)	0.474	meters
Reaction Time	1.077	seconds
Reaction Time at Stop	1.133	seconds
Margin for Overtaking (Mean)	5.530	seconds
Margin for Overtaking (Std. Dev.)	3.491	seconds
Gap (Mean)	0.584	meters
Gap (Std. Dev.)	0.247	meters

Table 8.1: Definitive Parameters ($\tilde{\mathbf{P}}^*$).

A quick analysis of these parameters shows particularities of the Swedish drivers in a highway. For instance, notice that they accept completely the limitations speed, they were set at 90 km/h and they maximum desired speed is around 100 km/h. Both Reaction Time and Reaction Time at Stop are quite similar which is coherent with the fact that the site is a highway and there is no congestion during the simulated interval and there are not the flow interruptions by signalized intersections that would occur at urban networks.

Figure 8.3 represents how far is $\tilde{\mathbf{P}}_k$ to reach the minimum $\tilde{\mathbf{P}}^*$. It was done by using the Euclidean distance $\left\|\tilde{\mathbf{P}}_k - \tilde{\mathbf{P}}^*\right\|$. As already mentioned, the algorithm approaches the value and, when arrives to the neighborhood, remains oscillating in the nearby.

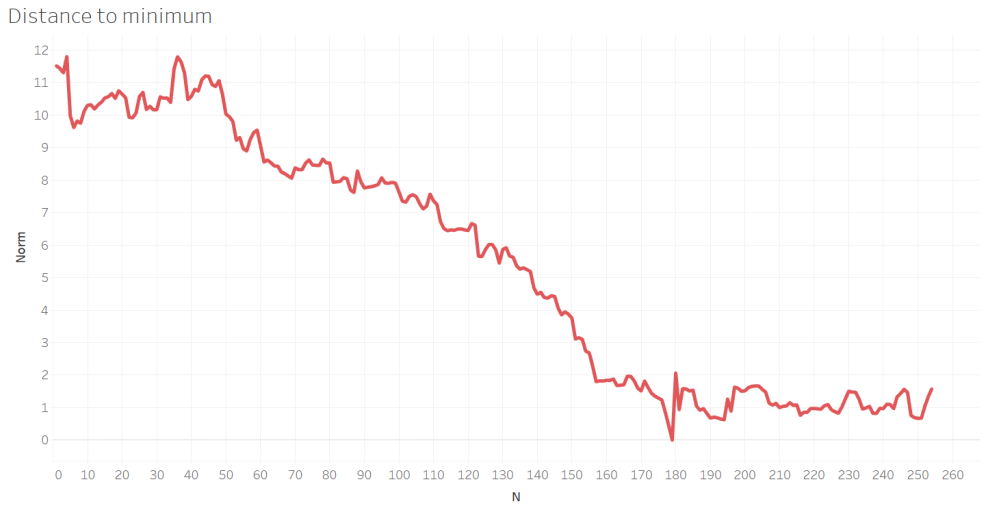


Figure 8.3: $\left\|\tilde{\mathbf{P}}_k - \tilde{\mathbf{P}}^*\right\|$ Evolution.

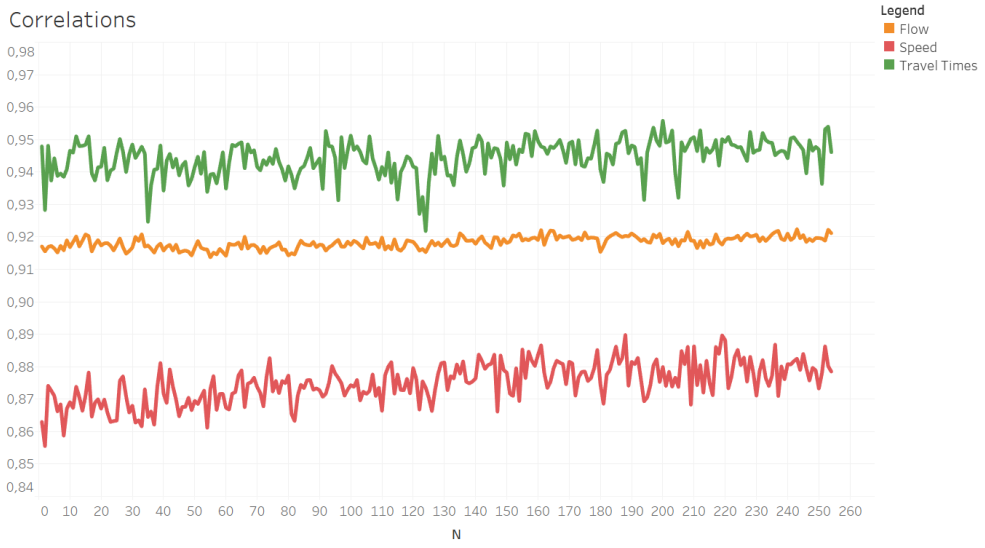


Figure 8.4: Correlation between simulated and observed data evolution.

Finally, Figure 8.4 plots the correlation between simulated and observed data evolution during the SPSA performance. They are almost stable during the performance. Some little oscillations, due to the random perturbations, and a little positive trend can be appreciated.

8.2 Calibration & Validation Results

This section will analyse the results of the model Calibration and Validation. The same analysis will be done for both results, hence, different used goodness-of-fit statistics will be explained.

8.2.1 Goodness-of-fit Measures

Normalised Root Mean Square Error (NRMSE)

Since it is the one used in the Objective Function, it will be computed for the three measures, flow, speed and travel times. It is easily computed as in Equation 4.1:

$$\mathrm{NRMSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\sqrt{\frac{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2}{n}}}{\max(\mathbf{x}) - \min(\mathbf{x})}$$

where $\mathbf{x}$ is a vector with the observed data and $\hat{\mathbf{x}}$, the correspondent simulated data. The lower it is the value, the better is the model.

Linear Regression R^2

The R^2 coefficient is used in linear models and represents the proportion of the variance in the response variable explained by the covariates.

In this work will be used as a measure of correlation between observed and simulated data by adjusting a linear model between them.

Theil's Inequality Coefficients

The Theil's coefficient is another interesting measure of error. It provides a measure on how close two time series are. Taking into account that the measurements of the traffic variables, as well as their replication by the simulation model area aggregated at the same regular time intervals, they can be considered as two time series, the

original one and its replication, and therefore Theil's coefficients can be used to estimate how close they are, [3, 10].

$$U(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\hat{x}_i^2} + \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}}$$

It is a number between $[0, 1]$ and the lower it is, the better is the simulated data.

The three other Theil's coefficients are proportions and can be calculated with the formulas written below:

- **Bias Proportion**: It measures how far is the simulated mean from the observed mean.

$$U_M(\mathbf{x}, \hat{\mathbf{x}}) = \frac{n(\hat{\mu} - \mu)^2}{\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}$$

- **Variance Proportion**: It measures how far is the simulated variance from the observed variance.

$$U_S(\mathbf{x}, \hat{\mathbf{x}}) = \frac{n(\hat{\sigma} - \sigma)^2}{\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}$$

- **Variance Proportion**: It measures the simulation errors.

$$U_C(\mathbf{x}, \hat{\mathbf{x}}) = \frac{2(1-\rho)n\hat{\sigma}\sigma}{\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}$$

where μ, σ are the mean and the variance of the observed data, $\hat{\mu}, \hat{\sigma}$ are the mean and the variance of the simulated data, n the number of observations and ρ is the correlation between them.

These three measures sum up to 1, since they are proportions. A good performance of the model should present $U_M, U_S \downarrow 0$ and $U_C \uparrow 1$, which implicates there is no bias and all they are fully correlated.

8.2.2 Calibration Results

In Table 8.2, all the goodness-of-fit measures are printed. Note the present bias in simulated flow with respect to the real flow, so as to in simulated speeds. Despite this fact, indicators show good simulated values in overall, with a NRMSE values around 10%.

	NRMSE	R^2	U	U_M	U_S	U_C
Flow	0.1148	0.8432	0.0938	0.3001	0.0524	0.6524
Speed	0.1039	0.7754	0.0324	0.1976	0.0035	0.8045
Travel Times	0.1082	0.9081	0.0444	0.0346	0.1677	0.8141
Mean	0.1090	0.8422	0.0569	0.1774	0.0745	0.7570

Table 8.2: Calibration Goodness-of-fit Measures

Flow Analysis

The bias in simulated flow can be further analysed regarding to Figures 8.5 and 8.6. As one can see in Figure 8.5, most of the simulated flows are below the graph bisector, which indicates that they are undervalued despite they are highly correlated.

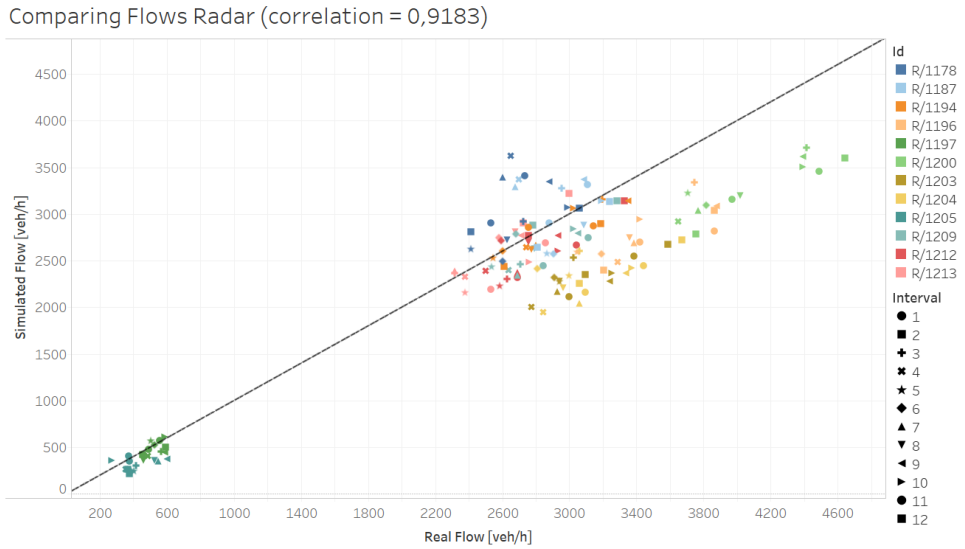


Figure 8.5: Linear Regression of Simulated Flow.

In Figure 8.6, the simulated and real flows, in blue and orange respectively, are plotted for the three first Radar sensors, R/1212, R/1209 and R/1204. Note that discrepancies between the values increase as one goes forward in the section.

Note also, in Figure 8.5 that there are two sensors with less quantity of vehicles. These two correspond to the exit lanes, as plotted in the map, in Figure 5.1.

Speed Analysis

The linear regression between real and simulated speed at Radars is plotted in Figure 8.7. In this case, the bias, less than in simulated flows, is in the other direction; simulated speeds are, in general, higher than the observed values.

The highest difference are in Radars R/1194 and R/1196, the ones in the middle of the section, where there is an entrance which was not in the model, as explained

in Section 5.3.2, which can be the reason because vehicles do not reduce the speed because they do not meet the cars coming from this input.

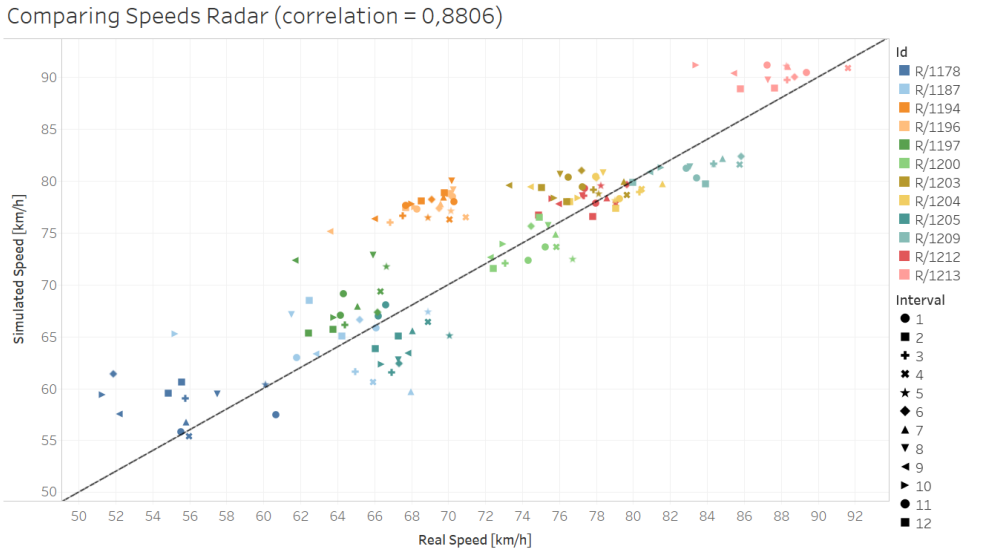


Figure 8.7: Linear Regression of Simulated Speed.

Travel Times Analysis

Travel times are highly correlated and don't present the high bias, as other magnitudes do, but they present more variance than the real values. In fact, it is easily detectable inn Figure 8.8, where the points are in clusters by pairs and far from the bisector.

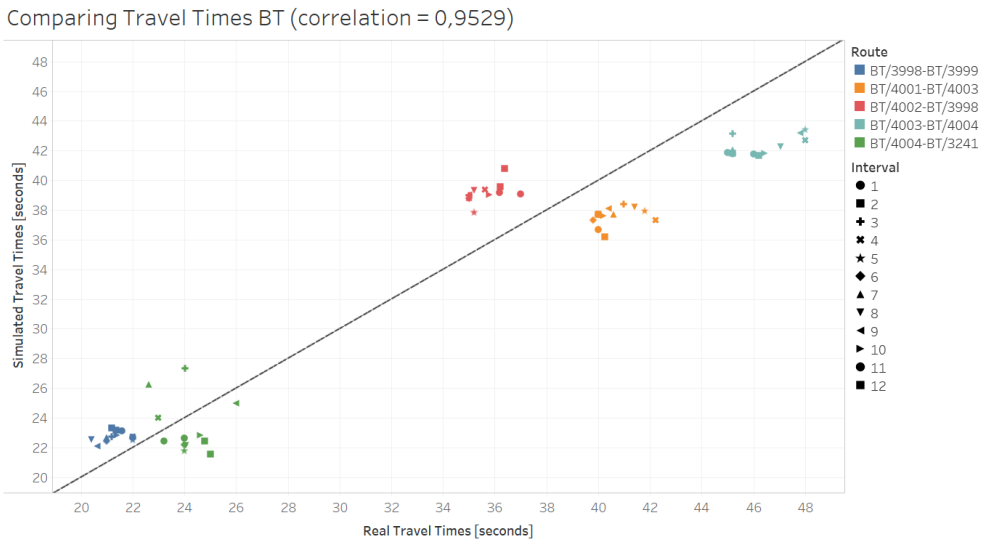


Figure 8.8: Linear Regression of Simulated Travel Times.

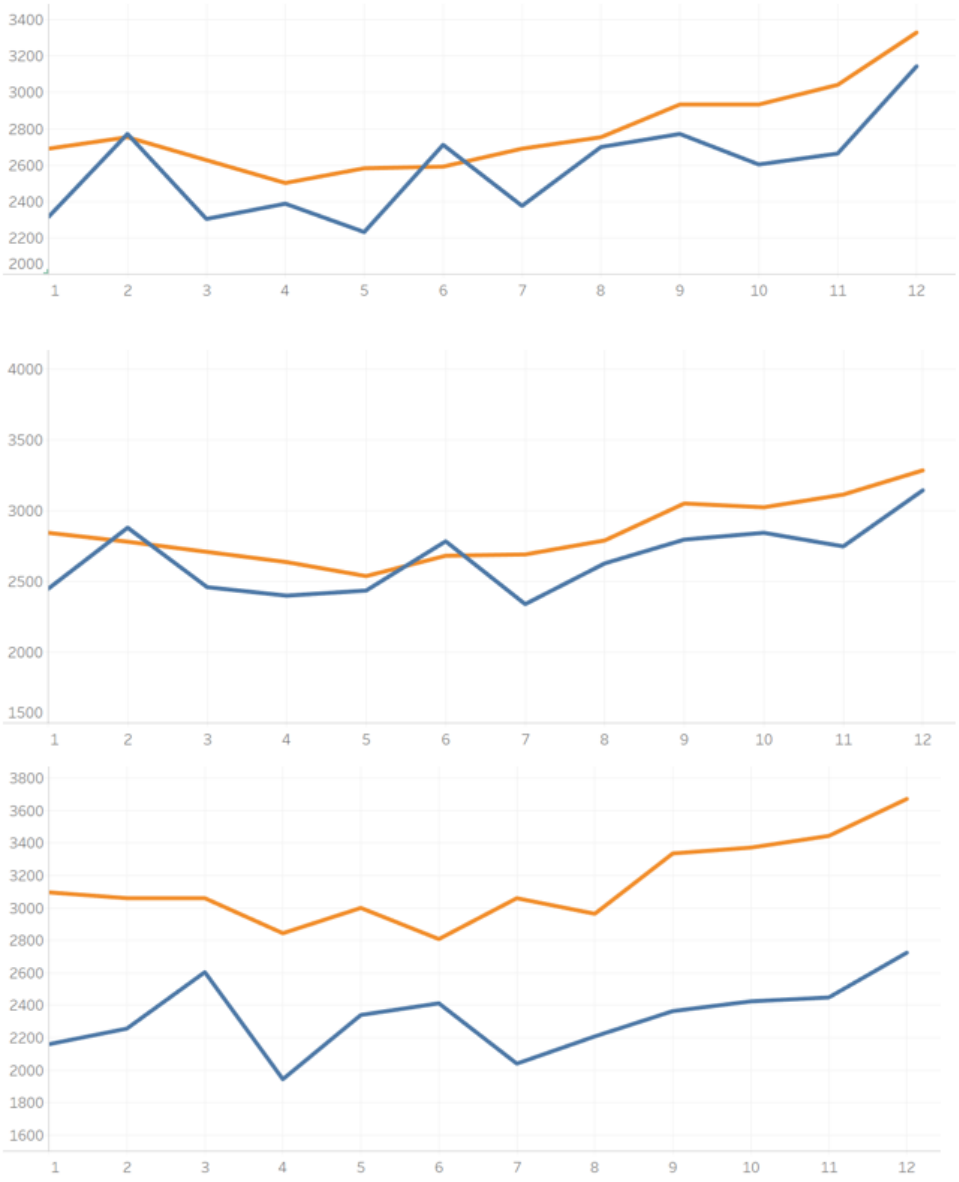


Figure 8.6: Simulated and Real Flows through time for sensors R/1212, R/1209 and R/1204.

8.2.3 Validation Results

As explained, validation of the model was made by using the parameters found during Calibration, $\mathbf{P}^*$, with another day data, 1 week later.

The goodness-of-fit values, Table 8.3, are quite worse than in calibration process, as usual. Even though, results show a good fit of the model to reality, showing NRMSE values around 12% and high correlations between simulated and observed values, around 84% in mean.

	NRMSE	R^2	U	U_M	U_S	U_C
Flow	0.1228	0.8324	0.1105	0.4279	0.0657	0.5105
Speed	0.1190	0.7983	0.0272	0.0010	0.0465	0.9595
Travel Times	0.1132	0.8939	0.0438	0.0096	0.0599	0.9474
Mean	0.1184	0.8415	0.0605	0.1461	0.0574	0.8058

Table 8.3: Validation Goodness-of-fit Measures

Note that, even the correlation is very high, flows have the worst indicators in overall. In that case, they present both bias and variance discrepancies. This will be further analysed trying to understand what is needed to improve the model.

Flow Analysis

Statistics measured in Table 8.3 show the worst fitting in flow, presenting high bias and variance. A further analysis shows discrepancies in the middle Radars, where the absence of some entrances in the model provokes that in R/1200, observed flow values are around 4200 veh/h and simulated values are around 3000 veh/h, as can be seen in Figure 8.9.

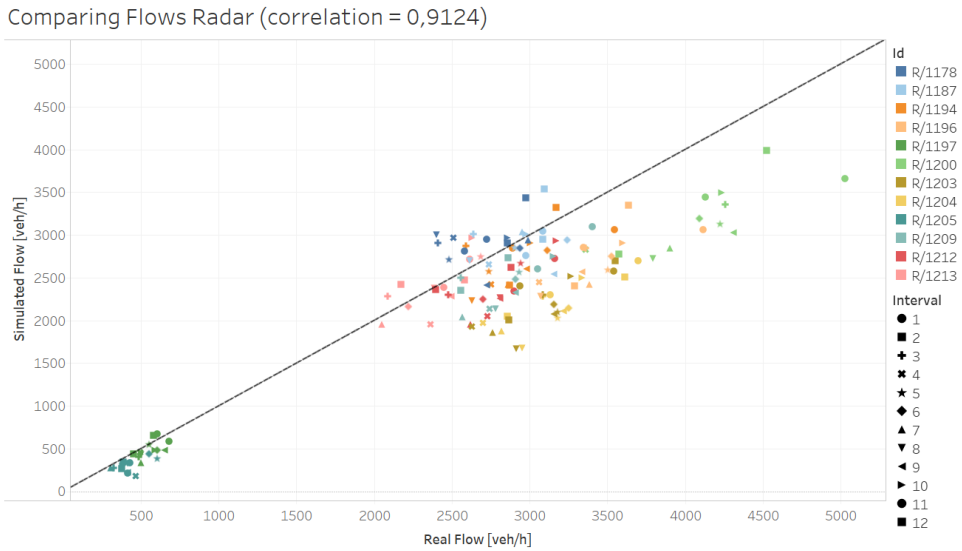


Figure 8.9: Linear Regression of Simulated Flow.

Speed Analysis

Differently from the calibration, speed does not present bias and correlation increases. Figure 8.10 shows a good adjustment between simulated and observed speeds. As before, worst adjustments are those in the middle of the section, where the structural model does not reflect totally the real highway.

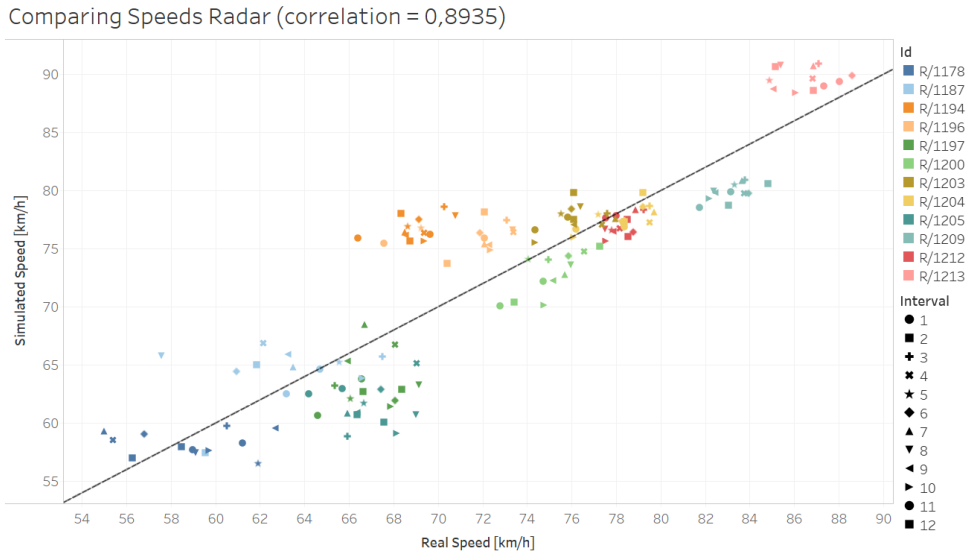


Figure 8.10: Linear Regression of Simulated Speed.

Travel Times Analysis

Results for Travel Times in the Validation step are also good. In this case no bias nor variance difference is detected and correlation between simulated and observed is high.

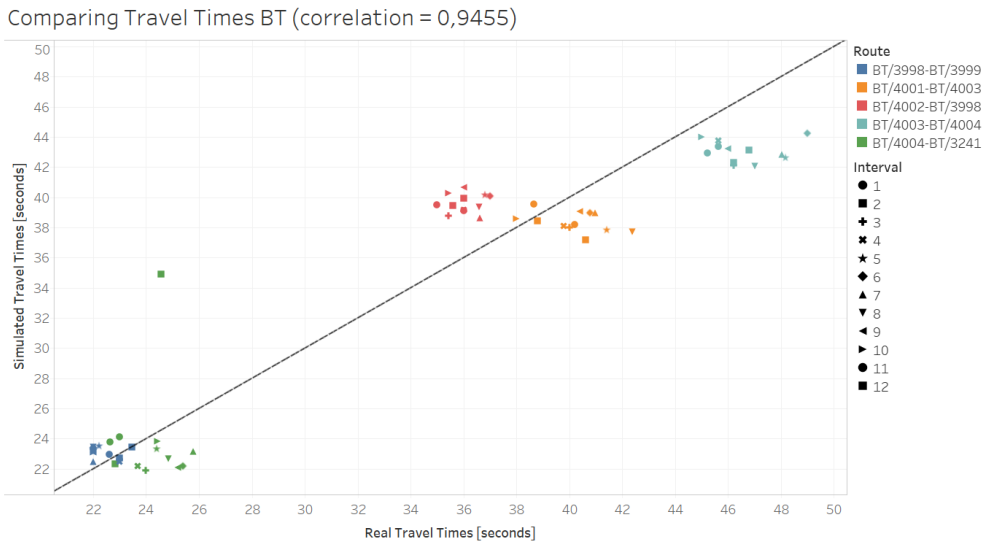


Figure 8.11: Linear Regression of Simulated Travel Times.

9
Conclusions

Modeling a complex system through simulation involves to consider a conceptual model of the system containing deterministic elements, such as the network description, and stochastic distributions for input variables. Data collected as measures of performance of the system have to be compared to resulting values obtained by simulation. Once the model is validated by comparison to real data, the model is useful to support *what if* questions for different scenarios. The validation exercise allows to predict different responses in different scenarios, adapting the input variables and the system configuration to show their effect to the system.

When the modeling exercise involves to reproduce the dynamic evolution of the system and considers explicitly stochastic elements appearing in the system, the technique is called simulation. Simulation is a powerful tool in supporting difficult decision making or in preventing different situations, for example.

A methodological framework has been stated in order to build valid models. A lot of knowledge has to be acquired to formulate some hypotheses under certain assumptions with the aim of building a computer model. One of the most important step in modeling is Calibration and Validation of the model and this has been the goal of this project. If one misses the step, then the model will be unable to capture the reality and will become useless, as *S. Maerivoet* and *B. de Moor* reflect in the quote:

> *—Whatever the modeling approach may be, researchers should always compare their results to the reality of the physical world. Ignoring this basic step, reduces the research in our opinion to nothing more than a mathematical exercise—*
>
> Sven Maerivoet and Bart de Moor, [12].

Simulation is especially useful for traffic modeling since traffic networks are complex systems with a lot of parameters and uncertainties involved and investments in the sector are usually expensive and affect directly to a wide range of people, drivers and pedestrians. Therefore, simulation of new configurations of the transportation network is crucial for decision making in Transportation Planning and Dynamic Traffic Management.

One of the most popular techniques to automatically assist the calibration of microscopic traffic simulation models is the Simulation-Based Optimization algorithm

called *Simultaneous Perturbation Stochastic Approximation* (SPSA), which requires a limited number of simulations. This method involves a non-differentiable optimization problem, solved using stochastic perturbations to progress to the minimum.

In this work, SPSA was applied to minimize discrepancies in three different observed measures simultaneously, flow, speed and travel times, and the objective function was properly chosen so that the same weight was given to the different measures. This minimization problem has been defined based on 12 different bounded behavioral parameters involved in the core models of microscopic traffic modeling. These parameters have different units and magnitudes.

Some problems arose when SPSA was applied to calibrate a microsimulation traffic model for a section of a Swedish highway supported with real data. These problems were analyzed and there were found the main causes, listed below:

- One of the most important point in Calibration and Validation is the quality of the data. Some inconsistencies have been found that may not be important for planning issues, where the level of aggregation softens the effects of them, but they are inadmissible when a valid traffic simulation model is desired.
- The selection of 12 different bounded parameters with different magnitudes lead the algorithm to a unusual performance, which has been fixed by modifying the classic SPSA algorithm, using normalized parameters and penalizing the objective function instead of considering hard restricted constraints through a Trust Region.

Innovations incorporated into the algorithm gave a calibrated model which has been validated with a new set of real data involving a similar traffic pattern. The new method implemented in this thesis opens a wide range of options for calibrating simultaneously different measures depending on different parameters with computational expensive simulations.

The procedure was applied to a linear motorway section not involving alternative paths, but the procedure can be extended to traffic network involving lots of OD paths by increasing the number of optimization variables including parameter related to Route-Choice modeling.

Parameter involving Car-Following models and driver behavior have been considered for the selected scenario, but the formulation of more complex optimization programs for calibration of traffic simulators affected by more complex modeling is a natural follow-up of the current work.

List of Figures

List of Tables

References

[1] https://en.wikipedia.org/wiki/European_route_E4, December 2016.

[2] https://en.wikipedia.org/wiki/Speed_limits_in_Sweden, December 2016.

[3] C. Antoniou, J. Barcelo, M. Brackstone, H. Celikoglu, B. Ciuffo, V. Punzo, P. Sykes, T. Toledo, P. Vortisch, and P. Wagner. Traffic simulation: Case for guidelines. *European Comission Joint Reasearch Centre - MULTITUDE*, 2014.

[4] C. Antoniou, V. Papathanasopoulou, and I. Markou. Online calibration for microscopic traffic simulation and dynamic multi-step prediction of traffic speed. *Transportation Research Part C*, pages 144–159, 2016.

[5] J. Barceló. *Fundamentals of Traffic Simulation*. Models, Traffic Models, Simulation and Traffic Simulation, Chapter 1. Springer, Barcelona, 2010. ISBN: 978-1-4419-6142-6.

[6] M. Bierlaire. Simulation and optimization: A short review. *Transportation Research*, pages 4–13, 2015.

[7] M. Bullejos, J. Barceló, and L. Montero. A due based bilevel optimization approach for the estimation of time sliced od matrices. *Elsevier, Ltd.*, pages 1–17, 2014.

[8] H. G. Daellebanch. *Simulation and Decisions Making: A Management Science Approach*. 1995.

[9] P. G. Gipps. A behavioural car-following model for computer simulation. *Pergamon Press, Ltd.*, 15–B(2):105–111, 1981.

[10] Y. Hollander and R. Liu. The principles of calibrating traffic microsimulation models. *Springer - Transportation*, (35):347–362, 2008.

[11] W. Huyer and A. Neumaier. Snobfit - stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software*, 35(2):1–25, 2008.

[12] S. Maerivoet and B. D. Moor. Traffic flow theory. 2005.

[13] C. Osorio and L. Chong. A computationally efficient simulation-based optimization algorithm for large-scale urban transportation problems. *Urban Transportation Problems*, 49(33):623–636, 2015.

[14] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[15] J. C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998.

[16] J. C. Spall. An overview of the simulation perturbation method for efficient optimization. *APL Technical Digest*, 19(4):482–492, 1998.

[17] S. L. Transport Simulation Systems. *Aimsun 8.1.3 Dynamic Simulators User's Manual*. 2016.

[18] I.-J. Wang and J. Spall. Stochastic optimisation with inequality constraints using simultaneous perturbations and penalty functions. *International Journal of Control*, pages 1232–1238, 2008.

[19] H. M. Zhang, J. Ma, and H. Dong. Calibration of micro simulation with heuristic optimization models. *ATRB Committee on Traffic Flow Theory*, pages 1–18, 2007.

[20] H. M. Zhang, J. Ma, and H. Dong. *Developing Calibration Tools for Microscopic Traffic Simulation Final Report Part II: Calibration Framework and Calibration of Local/-Global Driving Behavior and Departure/Route Choice Model Parameters*. California Partners for Advanced Transit and Highways, Davis, CA 95616, 2008.

Appendices

A Auxiliary Tables

A.1 Correspondences between Real Network and *AIMSUN*

	Type	Real ID	Latitude	Longitude	kmLabel	Dist. to 1st Sensor	Road	*AIMSUN*
1	R	1213	59.399	17.977	65.420	0.000	Primary	68123
2	BT	4002	59.394	17.987	64.970	0.450	Primary	68127
3	R	1212	59.394	17.988	64.970	0.450	Primary	66844
4	R	1209	59.390	17.997	64.265	1.155	Primary	66843
5	BT	3998	59.389	18.000	64.090	1.330	Primary	68128
6	BT	3999	59.385	18.005	63.580	1.840	Primary	68129
7	BT	4000	59.381	18.010	63.040	2.380	Primary	68130
8	R	1205	59.381	18.010	63.025	2.395	Out	68124
9	R	1204	59.381	18.010	63.025	2.395	Primary	66839
10	R	1203	59.379	18.012	62.805	2.615	Primary	68125
11	BT	4001	59.373	18.018	62.220	3.200	Primary	68131
12	R	1200	59.372	18.019	61.890	3.530	Primary	66837
13	BT	4003	59.366	18.021	61.395	4.025	Primary	68132
14	R	1196	59.367	18.021	61.330	4.090	Primary	66861
15	R	1197	59.367	18.020	61.330	4.090	Out	66862
16	R	1194	59.365	18.023	61.035	4.385	Primary	68126
17	BT	4004	59.362	18.026	60.645	4.775	Primary	68133
18	BT	3241	59.357	18.032	60.060	5.360	Primary	68134
19	R	1187	59.356	18.034	59.835	5.585	Primary	66831
20	R	1178	59.349	18.035	58.895	6.525	Primary	66825

Table A.1: Network Correspondences between Real ID and *AIMSUN*.

B
MATLAB Codes

B.1 SPSA function

This function launches the SPSA Algorithm, saving results and values in many files.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   TFM 2016                    %
%            Code: SPSA.m                       %
%            Author: Xavier Ros Roca            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ Pk, ofval] = SPSA( e, M , TRbool, seed)


rng(seed);
fileID = fopen('R.txt','w');
% fprintf(fileID, datestr(now,'dd-mm-yyyy HH:MM:SS'));
fprintf(fileID,'');
fclose(fileID);

fileID = fopen('P.txt','w');
% fprintf(fileID, datestr(now,'dd-mm-yyyy HH:MM:SS'));
fprintf(fileID,'');
fclose(fileID);

fileID = fopen('ofval.txt','w');
% fprintf(fileID, datestr(now,'dd-mm-yyyy HH:MM:SS'));
fprintf(fileID,'');
fclose(fileID);

fprintf('SPSA execution \n\n');
fprintf('Max Iterations: %d \n', M);
fprintf('Relative Error: %d \n', e);
fprintf('TrustRegion = %d\n', TRbool);


TRmin = [85,   2,  0.7,  0.2,    0.6,  0.3,  0.5, 0.7,  2,   1,   0.2,
    0.2];
TRmax = [120, 10,  1.4,  1.0,    2.0,  1.0,  1.8,  2.2,   10,  4,   1.5,
    0.8];

Pk = (3*TRmax+2*TRmin)/5 ;

k = 0;
stop = 0;
```

```
a = 2.2;
A = 6;
alfa = 0.602;


c =  0.6;
gamma = 0.101;

r = 1;

ofval = [];

fileID = fopen('logfile.txt','w');
fprintf(fileID, datestr(now,'dd-mm-yyyy HH:MM:SS'));
fprintf(fileID, '\n');
fprintf(fileID, 'Values of parameters: a = %f, c = %f, A = %d, r = %f', a, c,
     A, r);
fprintf(fileID, '\nTRmin:\n');
fprintf(fileID, '%f\t', TRmin);
fprintf(fileID, '\nTRmax:\n');
fprintf(fileID, '%f\t', TRmax);
fprintf(fileID, '\n');
fprintf(fileID, 'seed = %d\n', seed);
fclose(fileID);

while stop == 0
    k = k + 1;
    fprintf('-------------------------------------------------------\n');
    fprintf(datestr(now,'dd-mm-yyyy HH:MM:SS'));
    fprintf('\n');
    fprintf('Iteration: %d \n', k);

    fileID_P = fopen('P.txt','a');
    fprintf(fileID_P, '%f\t',Pk);
    fprintf(fileID_P, '\n');
    fclose(fileID_P);

    ck = c/(k+1)^gamma;

    Vk = (2*round(rand(12,1))-1)'; % Bernoulli. 1 or -1. Random direction.

    Pnk = Normalize(Pk, TRmin, TRmax);


    Pnka = Pnk + ck*Vk;

    Pka = InvNormalize(Pnka, TRmin, TRmax);
    Pka  = TR(Pka, TRmin, TRmax, TRbool);


    fprintf('FIRST SIMULATION STARTING... \n')
    [ofb Rb]= OF(Pk, TRmin, TRmax);

    fprintf('SECOND SIMULATION STARTING... \n')
    [ofa Ra] = OF(Pka, TRmin, TRmax);
    ofa
    ofb


    ofval = [ofval,ofb];
```

```
    R = Rb;


    fileID_R = fopen('R.txt','a');
    fprintf(fileID_R, '%f\t',R);
    fprintf(fileID_R, '\n');
    fclose(fileID_R);

    Rb = mean(Rb);

    fprintf('Mean of Correlations: %f\n', Rb );
    fprintf('Objective function value: %f\n', ofb);

    Gk = (ofa - ofb)./(ck*Vk);

    rk = r/(k^0.1);

    ak = a/(A+k+1)^alfa;

    Pnkk = Pnk - ak.*Gk  - ak*rk.*(max(Pnk-10,0)  + 3*min(Pnk, 0) +  3*max(Pk
    (7)-Pk(8), 0)*[0 0 0 0 0 0 +1 -1 0 0 0 0]);


    Pkk = InvNormalize(Pnkk, TRmin, TRmax);
    Pkk = TR(Pkk, TRmin, TRmax, TRbool);

    rel_grad = norm(Pnk - Pnkk)/norm(Pnkk);
    fprintf('Relative Error Gradient: %f\n', rel_grad);
    fprintf('------------------------------------------------------------\n')

    if rel_grad < e  % Rel. diff. is small enough to stop
        stop = 1;
    elseif k > M - 1
        stop = 2;
        fprintf('The algorithm did not converge in  %d  steps.\n', M);
    end

    Pk = Pkk;



    fileID_of = fopen('ofval.txt','a');
    fprintf(fileID_of, '%f\n',ofb);
    fclose(fileID_of);

    logfile(k, R, ofb, Pk, Gk, rel_grad);

end


er = norm(Pk - Pkk)/norm(Pkk);


resultsfile(k,stop,ofval,Pk,Gk, er);    % Printing the results in a .txt file

end
```

B.2 OF function

This function calls a function that launches the simulation.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    TFM 2016                    %
%            Code: OF.m                          %
%            Author: Xavier Ros Roca             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ of, R ] = OF( P , TRmin, TRmax)

load data;
desOut = [1,1,0,0]; % We want the GoF measured for FLOWS and SPEEDS
AIMSpath = 'C:\Program Files\TSS-Transport Simulation Systems\Aimsun 8.1/
    aconsole.exe';

[gofs,times,assMatr, Ry, Rt]=AIMSUN(ODPattern,AIMSpath,1,desOut,P);

of1 = sum(gofs(:,3));
of2 = mean(times(:,3));

Pn = Normalize(P, TRmin, TRmax);

of = (of1+of2) + 1/2*(sum(max(Pn-10,0).^2)  + 3*sum(min(Pn, 0).^2) + 3*max(P
    (7)-P(8), 0)^2);

R = [Ry Rt];

end
```

B.3 AIMSUN function

This code launches the Python code that runs a simulation with the parameters done.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    TFM 2016                    %
%            Code: AIMSUN.m                      %
%            Author: Xavier Ros Roca             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y,t,assMatr, Ry, Rt]=AIMSUN(ODPattern,AIMSpath,OS,desOutputs,P)

load scenarioInfo/Origins.txt;
load scenarioInfo/Destinations.txt;
load scenarioInfo/BT.txt;
for j=1:size(ODPattern,2)

    m=reshape(ODPattern(:,j),length(Origins),length(Destinations));

    filename=strcat('m',num2str(j-1),'.txt');
    fid=fopen(filename,'w');
    fprintf(fid,'id\t');
    fprintf(fid,'%i\t',Destinations);
    fprintf(fid,'\n');
    fclose(fid);
    fid=fopen(filename,'a');
    for i=1:length(Origins)
        fprintf(fid,'%i\t',Origins(i));
```

```
        fprintf(fid,'%5.2f\t',m(i,:));
        fprintf(fid,'\n');
    end
    fclose(fid);
end

detecPath='scenarioInfo/detectors.txt';

fid=fopen('scenarioInfo/scenario.txt');
simData=textscan(fid,'%u %u %s %s %s %s');
replID=simData{1,1};
dbID=simData{1,2};
DBname=cell2mat(simData{1,3});
angName=cell2mat(simData{1,4});
pyPath=cell2mat(simData{1,5});
assMatrName=cell2mat(simData{1,6});

if OS==1
    %For Windows Users
    parafile(P);
    commandTerminal= horzcat('"',AIMSpath,'" -script "',cd,'/',pyPath,'" "',
    cd,'/',angName,'" ',num2str(replID),' ',num2str(dbID));
    system(commandTerminal);
    fprintf('SIMULATION ENDED \n')

else
    %For MAC/Unix Users
    commandTerminal= horzcat(AIMSpath,' -script ',cd,'/',pyPath,' ',cd,'/',
    angName,' ',num2str(replID),' ',num2str(dbID));
    fid = fopen('./batchAIMS.sh','w');
    fprintf(fid,'osascript -e ''tell application "Terminal"\n');
    fprintf(fid,'\t do script "%s"\n',commandTerminal);
    fprintf(fid,'end tell''\n');
    fclose(fid);
    !chmod 755 ./batchAIMS.sh
    !xattr -d com.apple.quarantine ./batchAIMS.sh
    system('./batchAIMS.sh');
    while exist('testnet.matrix','file')==0
    end
    fid = fopen('./batchAIMS.sh','w');
    fprintf(fid,'osascript -e ''tell application "Terminal" to quit''');
    fclose(fid);
    system('./batchAIMS.sh');
end


conn = sqlite(DBname);

detecID=load(detecPath);
sqlQuery='SELECT oid + 0.0 as oid2,  ent +0.0 as ent2, flow + 0.0 as flow2,
    speed + 0.0 as speed2, occupancy + 0.0 as occupancy2, density+0.0 as
    density2 FROM MIDETEC WHERE did = 10 and sid = 1 and ent <> 0 order by
    oid, ent;';
sqlQuery2='SELECT oid + 0.0 as oid2, ent + 0.0 as ent2, ttime FROM MISECT
    WHERE did = 10 and sid = 1 and ent <> 0 order by oid, ent;';

simTimes_all = [];
for i = 1:length(BT)
    sqlQuery2 = horzcat('select A.oid + 0.0 as start_point, B.oid + 0.0 as
    end_point, A.idveh + 0.0 as idveh, A.timedet + 0.0 as time_ini, B.
    timedet + 0.0 as time_fin, B.timedet - A.timedet as ttime from (select *
```

```
     from DETEQUIPVEH where oid = ', num2str(BT(i,1)),  ' and did = 10) A
    inner join (select * from DETEQUIPVEH where oid = ', num2str(BT(i,2)), '
     and did = 10) B on A.idveh = b.idveh');
    simTimes = cell2mat(fetch(conn, sqlQuery2));


    summ = 14400;

    for j = 1:length(simTimes)
        if simTimes(j,4) < 38100 + summ
            simTimes(j,7) = 1;
        elseif simTimes(j,4) < 38400 + summ
            simTimes(j,7) = 2;
        elseif simTimes(j,4) < 38700 + summ
            simTimes(j,7) = 3;
        elseif simTimes(j,4) < 39000 + summ
            simTimes(j,7) = 4;
        elseif simTimes(j,4) < 39300 + summ
            simTimes(j,7) = 5;
        elseif simTimes(j,4) < 39600 + summ
            simTimes(j,7) = 6;
        elseif simTimes(j,4) < 39900 + summ
            simTimes(j,7) = 7;
        elseif simTimes(j,4) < 40200 + summ
            simTimes(j,7) = 8;
        elseif simTimes(j,4) < 40500 + summ
            simTimes(j,7) = 9;
        elseif simTimes(j,4) < 40800 + summ
            simTimes(j,7) = 10;
        elseif simTimes(j,4) < 41100 + summ
            simTimes(j,7) = 11;
        else
            simTimes(j,7) = 12;
        end
    end
    simTimes_all = [simTimes_all;simTimes];
end

simTimes_DT = array2table(simTimes_all(:,[1, 2, 7, 6]), 'VariableNames', {'
    Start', 'End', 'Int', 'ttime'});

simTimes = table2array(grpstats(simTimes_DT, {'Start', 'End', 'Int'}, {'min',
     'max', 'mean', 'median'}));
simTimes = sortrows(simTimes, [1, 2, 3]);




out=cell2mat(fetch(conn, sqlQuery));
outputs=[];
for i=1:length(detecID)
    outputs=[outputs; out(find(out(:,1)==detecID(i)),:)];
end

cols = [1, 2 , find(desOutputs == 1)+2];
simData = outputs(:, cols);

simData(find(simData==-1))=0;

load('scenarioInfo/trueData.txt');
load('scenarioInfo/trueTimes.txt');
```

```
assMatr=[];
% GoF measures are evaluated for the travel times
[t Rt] = fObjT(trueTimes,simTimes);


% The 12 GoF measures are then evaluated on the desired outputs
[y Ry] = (fObj(trueData,simData));


end
```

B.4 fObj and fObjT functions

This codes measure several Goodness-of-Fit values for different values.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   TFM 2016                    %
%            Code: fObj.m                       %
%            Author: Xavier Ros Roca            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y, R]=fObj(trueMoP,allMoP)


kk=size(trueMoP);
trueMoP = trueMoP(:,3:kk(2));
allMoP = allMoP(:,3:kk(2));

kk=size(trueMoP);
RMSE=[];
SE=[];
MAE=[];
NRMSE =[];
NMAE =[];
U =[];
ME = [];
NME = [];
% d=[]
RMSNE=zeros(1, kk(2));
GEH1=zeros(1, kk(2));
MANE=zeros(1, kk(2));
MNE=zeros(1, kk(2));
U1=[];
U2=[];



for i=1:kk(2)
    RMSE(i)=sqrt(sum(allMoP(:,i)-trueMoP(:,i)).^2/kk(1));
    SE(i)=sum(allMoP(:,i)-trueMoP(:,i)).^2;
    MAE(i)=sum(abs(allMoP(:,i)-trueMoP(:,i)))/kk(1);
    U1(i)=sqrt(sum((allMoP(:,i)).^2)/kk(1));
    U2(i)=sqrt(sum((trueMoP(:,i)).^2)/kk(1));
    U(:,i)=RMSE(i)/(U1(i)+U2(i));
    ME(i)=sum((allMoP(:,i)-trueMoP(:,i)))/(kk(1));
    NME(i)=(sum(allMoP(:,i)-trueMoP(:,i)))/(sum(trueMoP(:,i)));
```

```
    NRMSE(i)=100*sqrt(sum((allMoP(:,i)-trueMoP(:,i)).^2)/kk(1))/(max(trueMoP
    (:,i))-min(trueMoP(:,i)));
    NMAE(i)=sum(abs(allMoP(:,i)-trueMoP(:,i)))/(sum(abs(trueMoP(:,i))));

    for j=1:kk(1)
        if trueMoP(j,i)>0
            RMSNE(i)=RMSNE(i)+((allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i))^2;
            MANE(i)=MANE(i)+abs((allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i));
            MNE(i)=MNE(i)+(allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i);
        elseif allMoP(j,i)>0
            RMSNE(i)=RMSNE(i)+1;
            MANE(i)=MANE(i)+1;
            MNE(i)=MNE(i)+1;
        end
        if trueMoP(j,i)>0 || allMoP(j,i)>0
            GEH=sqrt(2*(allMoP(j,i)-trueMoP(j,i))^2/(allMoP(j,i)+trueMoP(j,i)
    ));
            if GEH<=1
                GEH1(i)=GEH1(i)+1;
            end
        else
            GEH1(i)=GEH1(i)+1;
        end
    end

    MNE(i)=MNE(i)/kk(1);
    RMSNE(i)=sqrt(RMSNE(:,i)/kk(1));
    MANE(i)=MANE(i)/kk(1);
    GEH1(i)=(kk(1)-GEH1(i))/kk(1);
end

R1 = corrcoef(trueMoP(:,1),allMoP(:,1));
R2 = corrcoef(trueMoP(:,2),allMoP(:,2));
R = [R1(1,2), R2(1,2)];




y=[RMSE;RMSNE;NRMSE;GEH1;MAE;MANE;NMAE;SE;U;ME;MNE;NME]';

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   TFM 2016                    %
%            Code: fObjT.m                      %
%            Author: Xavier Ros Roca            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [t, R] =fObjT(trueMoP,allMoP)



kk=size(trueMoP);

trueMoP = trueMoP(:, 7);
allMoP = allMoP(:, 7);

kk=size(trueMoP);
RMSE=[];
SE=[];
MAE=[];
NRMSE =[];
```

```
NMAE =[];
U =[];
ME = [];
NME = [];
RMSNE=zeros(1, kk(2));
GEH1=zeros(1, kk(2));
MANE=zeros(1, kk(2));
MNE=zeros(1, kk(2));
U1=[];
U2=[];

for i=1:kk(2)
    RMSE(i)=sqrt(sum(allMoP(:,i)-trueMoP(:,i)).^2/kk(1));
    SE(i)=sum(allMoP(:,i)-trueMoP(:,i)).^2;
    MAE(i)=sum(abs(allMoP(:,i)-trueMoP(:,i)))/kk(1);
    U1(i)=sqrt(sum((allMoP(:,i)).^2)/kk(1));
    U2(i)=sqrt(sum((trueMoP(:,i)).^2)/kk(1));
    U(:,i)=RMSE(i)/(U1(i)+U2(i));
    ME(i)=sum((allMoP(:,i)-trueMoP(:,i)))/(kk(1));
    NME(i)=(sum(allMoP(:,i)-trueMoP(:,i)))/(sum(trueMoP(:,i)));
    NRMSE(i)=100*sqrt(sum((allMoP(:,i)-trueMoP(:,i)).^2)/kk(1))/(max(trueMoP
    (:,i))-min(trueMoP(:,i)));
    NMAE(i)=sum(abs(allMoP(:,i)-trueMoP(:,i)))/(sum(abs(trueMoP(:,i))));



    for j=1:kk(1)
        if trueMoP(j,i)>0
            RMSNE(i)=RMSNE(i)+((allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i))^2;
            MANE(i)=MANE(i)+abs((allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i));
            MNE(i)=MNE(i)+(allMoP(j,i)-trueMoP(j,i))/trueMoP(j,i);
        elseif allMoP(j,i)>0
            RMSNE(i)=RMSNE(i)+1;
            MANE(i)=MANE(i)+1;
            MNE(i)=MNE(i)+1;
        end
        if trueMoP(j,i)>0 || allMoP(j,i)>0
            GEH=sqrt(2*(allMoP(j,i)-trueMoP(j,i))^2/(allMoP(j,i)+trueMoP(j,i)
    ));
            if GEH<=1
                GEH1(i)=GEH1(i)+1;
            end
        else
            GEH1(i)=GEH1(i)+1;
        end
    end

    MNE(i)=MNE(i)/kk(1);
    RMSNE(i)=sqrt(RMSNE(:,i)/kk(1));
    MANE(i)=MANE(i)/kk(1);
    GEH1(i)=(kk(1)-GEH1(i))/kk(1);
end
R1 = corrcoef(trueMoP(:,1),allMoP(:,1));

R = R1(1,2);

t=[RMSE;RMSNE;NRMSE;GEH1;MAE;MANE;NMAE;SE;U;ME;MNE;NME]';

end
```

B.5 Normalize and InvNormalize functions

This functions normalize the parameters.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    TFM 2016                    %
%            Code: Normalize.m                   %
%            Author: Xavier Ros Roca             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  Pn  = Normalize( P , TRmin, TRmax)

Pn = 10*(P-TRmin)./(TRmax-TRmin);

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    TFM 2016                    %
%            Code: InvNormalize.m                %
%            Author: Xavier Ros Roca             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  P  = InvNormalize( Pn , TRmin, TRmax)

P = TRmin + Pn.*(TRmax-TRmin)/10;

end
```

C
Python Codes

C.1 Traffic State Creation

This code is executed inside *AIMSUN* environment to generate the different Traffic Demand States from Real Radar Data.

```
#####################################
#             TFM 2016              #
#    Code: script_traffic_states.py  #
#    Autor: Xavier Ros Roca          #
#####################################
trafficStateFile='C:\\Users\\xavier.ros.roca\\Desktop\\Others\\TFM\\02_Data\\
    traffic_state_data_flow_19_march.csv'
turnsFile='C:\\Users\\xavier.ros.roca\\Desktop\\Others\\TFM\\02_Data\\
    traffic_state_data_turns_19_march.csv'

def findSection( model, entry ):
  section = model.getCatalog().find( int(entry) )
  if section.isA( "GKSection" ) == False:
    section = None
  return section

def getStateFolder( model ):
  folderName = "GKModel::trafficStates"
  folder = model.getCreateRootFolder().findFolder( folderName )
  if folder == None:
    folder = GKSystem.getSystem().createFolder( model.getCreateRootFolder(),
    folderName )
  return folder

def createState( model, name,fromTime,durationTime):
  state = GKSystem.getSystem().newObject( "GKTrafficState", model )
  state.setName( name )
  state.setFrom(QTime.fromString(fromTime, Qt.ISODate))
  state.setDuration(GKTimeDuration.fromString( durationTime))
  vehicle = model.getCatalog().find(53)
  state.setVehicle( vehicle )
  folder = getStateFolder( model )
  folder.append( state )
  return state

def setEntranceFlow(model,state,sectionName,flow):
  section=findSection(model,sectionName)
  state.setEntranceFlow(section,None,float(flow))
```

```
def setTurns(model,state,originName,destinationName,percentage):
  origin=findSection(model,originName)
  destination=findSection(model,destinationName)
  state.setTurningPercentage(origin,destination,None,float(percentage))

#First a dictionary with the turns info is created
dictTurns={}
for line in open(turnsFile,'r').readlines():
  tokens2=line.split(",")
  dictTurns[tokens2[0]] = line


for line in open( trafficStateFile, "r" ).readlines():
  tokens = line.split(",")
  state = createState( model, tokens[0],tokens[0],"0:05:00")
  setEntranceFlow(model,state,tokens[1],tokens[2])
  setEntranceFlow(model,state,tokens[3],tokens[4])
  setEntranceFlow(model,state,tokens[5],tokens[6])
  setEntranceFlow(model,state,tokens[7],tokens[8])

  turnsLine = dictTurns[tokens[0]]
  turns=turnsLine.split(",")
  setTurns(model,state,turns[1],turns[2],turns[3])
  setTurns(model,state,turns[4],turns[5],turns[6])
  setTurns(model,state,turns[7],turns[8],turns[9])
  setTurns(model,state,turns[10],turns[11],turns[12])
  setTurns(model,state,turns[13],turns[14],turns[15])
  setTurns(model,state,turns[16],turns[17],turns[18])

model.getCommander().addCommand( None )
print "Done"
```

C.2 *AIMSUN* executor

This code is executed from *MATLAB* and executes the microscopic simulation in *AIMSUN* from Batch.

```
################################
#             TFM 2016         #
#    Code: odSimExecutor3.py   #
#    Autor: Xavier Ros Roca    #
################################
import sys
import os
import time

from PyANGBasic import *
from PyANGKernel import *
from PyANGConsole import *
from PyANGAimsun import *
from PyMesoPlugin import *

def simulateMeso(model, rep, assignMatrixFileName):
    global fileLog
    experiment = rep.getExperiment()
    fileLog.write("Simulating meso experiment %s...\n" % experiment.getName()
    )
```

```
    fileLog.flush()
    plugin = GKSystem.getSystem().getPlugin( "AMesoPlugin" ) # AMesoPlugin
    simulator=AMesoDTASimulator()  # AMesoDTASimulator
    simulator.setModel( model )

    task=GKSimulationTask()
    task.replication=rep
    if experiment.getEngineMode()==GKExperiment.eOneShot:
        task.mode=GKReplication.eBatch
    else:
        task.mode=GKReplication.eBatchIterative
    simulator.addSimulationTask( task )
    # simulator.setGatherProportions(True, assignMatrixFileName)
    simulator.simulate()

def simulateMicro(model, rep, assignMatrixFileName):
    plugin = GKSystem.getSystem().getPlugin("GGetram")
    simulator = plugin.getCreateSimulator(model)
    if not simulator.isBusy():
        if rep is not None and rep.isA("GKReplication"):
            if rep.getExperiment().getSimulatorEngine() == GKExperiment.
    eMicro:
                simulator.addSimulationTask(GKSimulationTask(rep,
    GKReplication.eBatch))
                simulator.simulate()


def main(argv):
    global fileLog

    if len(argv) != 4:
        print "Usage: aconsole -script %s ANG_FILE_NAME ID_REP ID_DATABASE" %
     argv[0]
        return -1

    angFileName = argv[1]


    params = []
    with open('C:\\Users\\xavier.ros.roca\\Desktop\\Others\\TFM\\05_Models\\
    Calibration\\parameters.txt', 'r') as file:
        for line in file:
            params = [float(val) for val in line.strip().split(' ')]

    #print params ####
    angAbsName = os.path.basename(angFileName)
    angName = os.path.splitext(angAbsName)[0]
    fileLogPath = os.path.dirname(angFileName) + os.sep+angName+'.log'


    assignMatrixFileName = os.path.dirname(angFileName) +os.sep+angName+'.
    matrix'

    if not os.path.exists(fileLogPath):
        fileLog = open( fileLogPath, "w" )
    else:
        fileLog = open( fileLogPath, "a" )

    idRep = int(argv[2])
    idDatabase = int(argv[3])
```

```
system=GKSystem()

fileLog.write("\nooooooooooooooooooooooooooooooooooooooooooooooooo\n" )
fileLog.write("Date: %s - " % time.asctime( time.localtime(time.time()) )
 )
fileLog.write("%s - CONSOLE - " % str(system.getAppVersion()) )
fileLog.write("ANG FILE: %s\n" %str(angFileName) )
fileLog.write("Id Replication/Result: %d\n" %idRep )
fileLog.write("Id Database: %d\n" %idDatabase )
fileLog.write("Assignment Matrix file: %s\n" %assignMatrixFileName )
fileLog.write("\nooooooooooooooooooooooooooooooooooooooooooooooooo\n" )

# Start a Console
console = ANGConsole()

fileLog.flush()

# Load a network
if console.open(angFileName):
    # Create a backup
    console.save(angFileName+".old")
    console.save(angFileName)
    model=console.getModel()

    fileLog.write( "-------------------------------------------\n")

    fileLog.write( "Network: %s \n" % str(model.getDocumentFileName()) )
    fileLog.write( "-------------------------------------------\n")
    fileLog.flush()

    vehicleType = model.getType("GKVehicle") ####
    carVeh = model.getCatalog().find(53) ####

    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
maxSpeedMean ", 1), params[0])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
maxSpeedDev", 1), params[1])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
speedAcceptanceMean", 1), params[2])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
speedAcceptanceDev", 1), params[3])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
minDistMean", 1), params[4])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
minDistDev", 1), params[5])
    reactionTimes = carVeh.getVariableReactionTimes()[0]
    reactionTimes.reactionTime = params[6]
    reactionTimes.reactionTimeAtStop = params[7]
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
overtakingMarginMean", 1), params[8])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
overtakingMarginDev", 1), params[9])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
minimunHeadwayMean", 1), params[10])
    carVeh.setDataValueDouble(vehicleType.getColumn("GKVehicle::
minimunHeadwayDev", 1), params[11])

    replication = model.getCatalog().find(idRep);
    if replication!=None and (replication.isA('GKReplication') or
replication.isA('GKExperimentResult')):
        replication.setDBId(idDatabase)
```

```
            simulateMicro(model, replication, assignMatrixFileName)

            fileLog.write( "Simulation Finished at %s\n"%time.asctime( time.
    localtime(time.time()) ))
            fileLog.flush()
        else:
            fileLog.write( "Cannot find the replication % d\n"%idRep)
            fileLog.flush()
            fileLog.write( "Cannot1")
        network = model.getDocumentFileName()  ####
        #console.save(network[:-4]+"_tmp.ang")

        console.close()
    else:
        fileLog.write( "Cannot load the network\n")
        fileLog.flush()
        console.getLog().addError( "Cannot load the network" )



try:
    sys.exit(main(sys.argv))
except:
    import traceback
    traceback.print_exc()
```

D
R codes

D.1 Radars Data Reader

This code reads the Real Data from Radars, cleans it and aggregates it to the desired level.

```
#####################################
#               TFM 2016            #
#     Code: reading_data_RADAR.R    #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)

##################

A <- data.frame(read.csv("02_Data/raw_data/sensordata_2015_03-2015_05.csv",
    sep=";", header = T, stringsAsFactors = FALSE ))
str(A)


#########################################################


A$Date <- as.Date(substr(A$timestamp, 1, 10))

D <- A[which(A$Date %in% c(as.Date("2015-03-21"),
                           as.Date("2015-03-28"),
                           as.Date("2015-03-19"),
                           as.Date("2015-03-26"))),]

D$Year <- as.numeric(substr(D$timestamp, 1, 4))
D$Month <- as.numeric(substr(D$timestamp, 6, 7))
D$Day <- as.numeric(substr(D$timestamp, 9, 10))
D$Hour <- as.numeric(substr(D$timestamp, 12, 13))
D$Minute <- as.numeric(substr(D$timestamp, 15, 16))
D$Date <- as.Date(substr(D$timestamp, 1, 10))
D$timestamp <- as.POSIXct(D$timestamp)

names(D)[which(names(D) == "timestamp")] <- "DateTime"
names(D)[which(names(D) == "sensorId")] <- "id"
```

```
names(D)[which(names(D) == "totalFlow")] <- "flow"

D$type <- "R"

D <- D %>%
  select(id, type, DateTime, Date, Hour, Minute, flow, speed)

D$speed <- 3.6*D$speed
D$id <- paste0("R/", D$id)

str(D)
D <- unique(D)

###### AGGREGATION ####

gap <- 5
u <- 0:59

int = c(0)

for(j in 1:(60/gap)) {
  v <- rep(gap*(j-1), gap)
  int <- c(int, v)
}

int <- int[-1]


DF_aux <- data.frame(Minute = u, Int = int)

#######

D$SF <- D$speed*D$flow

D3 <- D %>%
  left_join(DF_aux, by="Minute") %>%
  rename(index = Int) %>%
  group_by(id, type, Date, Hour, index) %>%
  summarise(DateTime = min(DateTime),
            flow = mean(flow, na.rm = T),
            flow_sum = sum(flow, na.rm=T),
            speed = mean(SF/flow_sum, na.rm=T) ) %>%
  rename(Minute = index)


D3$flow_sum <- NULL

D4 <- D3[which(D3$Hour %in% 6:19),]

D4 <- D4[,c(1:2, 6, 3:5, 7:8 )]


####### Fusion and writing CSV ##########

write.table(D4, "radar_data.csv", row.names = FALSE, sep=";", dec=",", quote=
    FALSE)
save.image("02_Data/Clean_data_16112016.RData")
```

D.2 Bluetooth Data Reader

This code reads the Real Data from Bluetooth, cleans it and aggregates it to the desired level.

```
#####################################
#               TFM 2016            #
#     Code: reading_data_BT.R       #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)

LF <- list.files(path="02_Data/raw_data/blip/")

LF

BT_df <- data.frame(Timestamp = NA ,
                    StartPoint = NA,
                    StartPointName = NA,
                    EndPoint = NA,
                    EndPointName = NA,
                    MinMeasuredTime = NA,
                    MaxMeasuredTime = NA,
                    AvgMeasuredTime = NA,
                    MedianMeasuredTime = NA,
                    SampleCount = NA,
                    AccuracyLevel = NA,
                    ConfidenceLevel = NA)

# i = 1
for(i in 1:length(LF)) {
  file <- paste0("02_Data/raw_data/blip/", LF[i])
  a <- data.frame(read.csv(file, header = T, sep=",", stringsAsFactors =
    FALSE))

  BT_df <- rbind(BT_df, a)

}

BT_df <- unique(BT_df)

BT_df <- BT_df[-1,]
names(BT_df)[1] <- "DateTime"
BT_df$StartPoint <- BT_df$EndPoint <- NULL
names(BT_df)[2] <- "StartPoint"
names(BT_df)[3] <- "EndPoint"


head(BT_df)
BT_df$DateTime <- str_replace_all(BT_df$DateTime, "mar", "03")
BT_df$DateTime <- str_replace_all(BT_df$DateTime, "apr", "04")
BT_df$DateTime <- str_replace_all(BT_df$DateTime, "maj", "05")


BT_df$Date <- as.Date(substr(BT_df$DateTime, 1, 10))
```

```
BT_df <- BT_df[which(BT_df$Date %in% c(as.Date("2015-03-21"), as.Date("
    2015-03-28"), as.Date("2015-03-19"), as.Date("2015-03-26"))),]

BT_df$StartPoint <- as.character(BT_df$StartPoint)
BT_df$EndPoint <- as.character(BT_df$EndPoint)

BT_df$Year <- as.numeric(substr(BT_df$DateTime, 1, 4))
BT_df$Month <- as.numeric(substr(BT_df$DateTime, 6, 7))
BT_df$Day <- as.numeric(substr(BT_df$DateTime, 9, 10))
BT_df$Hour <- as.numeric(substr(BT_df$DateTime, 12, 13))
BT_df$Minute <- as.numeric(substr(BT_df$DateTime, 15, 16))


BT_df$DateTime <- as.POSIXct(BT_df$DateTime, format= "%Y-%m-%d %H:%M:%S")
BT_df$Start_ID <- paste0("BT/", BT_df$StartPoint)
BT_df$End_ID <- paste0("BT/", BT_df$EndPoint)
summary(BT_df)

table(BT_df$StartPoint, BT_df$EndPoint)

loc <- read.csv("place.csv", sep=";", header=T, dec=",", stringsAsFactors =
    FALSE)


#### aggregation #####

gap <- 5
u <- 0:59

int = c(0)

for(j in 1:(60/gap)) {
  v <- rep(gap*(j-1), gap)
  int <- c(int, v)
}

int <- int[-1]

DF_aux <- data.frame(Minute = u, Int = int)

################

BT_df_agg <- BT_df %>%
  left_join(DF_aux, by="Minute") %>%
  rename(index = Int) %>%
  group_by(Start_ID, End_ID, Date, Hour, index) %>%
  summarise(DateTime = min(DateTime, na.rm=T),
            MinMeasuredTime = min(MinMeasuredTime, na.rm = T),
            MaxMeasuredTime = max(MaxMeasuredTime, na.rm = T),
            AvgMeasuredTime = sum(AvgMeasuredTime*SampleCount, na.rm=T)/sum(
    SampleCount, na.rm=T),
            MedianMeasuredTime = median(MedianMeasuredTime),
            SampleCount = sum(SampleCount) ) %>%
  rename(Minute = index) %>%
  data.frame()

BT_df_agg[which(is.na(BT_df_agg$AvgMeasuredTime)),]$AvgMeasuredTime <- 0

BT_df2 <- BT_df_agg

####### Fusion and writing CSV ##########
```

```
write.table(BT_df2, "BT_data.csv", row.names = FALSE, sep=";", dec=",", quote
    =FALSE)
save.image("02_Data/Clean_data_BT_16112016.RData")
```

D.3 Python Input Writer

This code prepares the input for the Python code written at Appendix C.1

```
#####################################
#               TFM 2016            #
#     Code: writing_python_input.R  #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)

source("01_Codes/reading_data_RADAR.R")

##### Input for the Real Data State ####

DAY <- as.Date("2015-03-19")

## 1. flow ###

# main flow (section 22727)
loc <- read.csv("place.csv", sep=";", header=T, dec=",", stringsAsFactors =
    FALSE)

Aux <- D4[which(D4$id == "R/1213" & D4$Date == DAY),c("DateTime", "flow")]

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$Section <- 22727

input_flow <- select(Aux, time, Section, flow)

# 1209-1212 (section 878)

Aux_1209 <- D4[which(D4$id == "R/1209" & D4$Date == DAY),c("DateTime", "flow"
    )]
Aux_1212 <- D4[which(D4$id == "R/1212" & D4$Date == DAY),c("DateTime", "flow"
    )]

Aux <- left_join(Aux_1212, Aux_1209, by = "DateTime")
Aux$flow <- round(Aux$flow.y - Aux$flow.x)
Aux$flow[which(Aux$flow < 0)] <- 0

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$Section <- 878

input_flow <- left_join(input_flow, select(Aux, time, Section, flow), by="
    time")


# 1200 - 1203 (section 23470)
```

```
Aux_1200 <- D4[which(D4$id == "R/1200" & D4$Date == DAY),c("DateTime", "flow"
    )]
Aux_1203 <- D4[which(D4$id == "R/1203" & D4$Date == DAY),c("DateTime", "flow"
    )]

Aux <- left_join(Aux_1203, Aux_1200, by = "DateTime")
Aux$flow <- round(Aux$flow.y - Aux$flow.x)
Aux$flow[which(Aux$flow < 0)] <- 0

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$Section <- 23470

input_flow <- left_join(input_flow, select(Aux, time, Section, flow), by="
    time")



# 1196 - 1194 (section 66855)

Aux_1196 <- D4[which(D4$id == "R/1196" & D4$Date == DAY),c("DateTime", "flow"
    )]
Aux_1194 <- D4[which(D4$id == "R/1194" & D4$Date == DAY),c("DateTime", "flow"
    )]

Aux <- left_join(Aux_1194, Aux_1196, by = "DateTime")
Aux$flow <- round(Aux$flow.y - Aux$flow.x)
Aux$flow[which(Aux$flow < 0)] <- 0

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$Section <- 66855

input_flow <- left_join(input_flow, select(Aux, time, Section, flow), by="
    time")


write.table(input_flow, "02_Data/traffic_state_data_flow_19_march.csv", quote
    =FALSE, sep=",", row.names=FALSE, col.names = FALSE)




############## Turns #################

### 1r 66982 = 66684 + 387
Aux  <- D4[which(D4$id == "R/1204" & D4$Date == DAY),c("DateTime", "flow")]
Aux2 <- D4[which(D4$id == "R/1203" & D4$Date == DAY),c("DateTime", "flow")]
Aux3 <- D4[which(D4$id == "R/1205" & D4$Date == DAY),c("DateTime", "flow")]

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$DateTime <- NULL
Aux2$time <- substr(Aux2$DateTime, 12, 20)
Aux2$DateTime <- NULL
Aux3$time <- substr(Aux3$DateTime, 12, 20)
Aux3$DateTime <- NULL

Aux <- full_join(Aux, Aux2, by="time")
Aux <- full_join(Aux, Aux3, by="time")
names(Aux)[4] <- "flow.z"

Aux$flow_sum <- Aux$flow.y + Aux$flow.z
```

```
Aux$T1 <- round((Aux$flow.y/Aux$flow_sum)*100)
Aux$T2 <- round((Aux$flow.z/Aux$flow_sum)*100)

Aux$Main <- Aux$MAIN <- 66982
Aux$Main2 <- 66684
Aux$Out <- 387

input_turns <- select(Aux, time, Main, Main2, T1, MAIN, Out, T2)


### 2n 9866 = 9483 + 2330
Aux  <- D4[which(D4$id == "R/1196" & D4$Date == DAY),c("DateTime", "flow")]
Aux2 <- D4[which(D4$id == "R/1194" & D4$Date == DAY),c("DateTime", "flow")]
Aux3 <- D4[which(D4$id == "R/1197" & D4$Date == DAY),c("DateTime", "flow")]

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$DateTime <- NULL
Aux2$time <- substr(Aux2$DateTime, 12, 20)
Aux2$DateTime <- NULL
Aux3$time <- substr(Aux3$DateTime, 12, 20)
Aux3$DateTime <- NULL

Aux <- full_join(Aux, Aux2, by="time")
Aux <- full_join(Aux, Aux3, by="time")
names(Aux)[4] <- "flow.z"

Aux$flow_sum <- Aux$flow.y + Aux$flow.z
Aux$T1 <- round((Aux$flow.y/Aux$flow_sum)*100)
Aux$T2 <- round((Aux$flow.z/Aux$flow_sum)*100)

Aux$Main <- Aux$MAIN <- 9866
Aux$Main2 <- 9483
Aux$Out <- 2330

input_turns <- left_join(input_turns, select(Aux, time, Main, Main2, T1, MAIN
    , Out, T2), by="time")



### 3r  66785 = 14487 + 66881
Aux  <- D4[which(D4$id == "R/1187" & D4$Date == DAY),c("DateTime", "flow")]
Aux2 <- D4[which(D4$id == "R/1178" & D4$Date == DAY),c("DateTime", "flow")]

Aux$time <- substr(Aux$DateTime, 12, 20)
Aux$DateTime <- NULL
Aux2$time <- substr(Aux2$DateTime, 12, 20)
Aux2$DateTime <- NULL


Aux <- full_join(Aux, Aux2, by="time")

Aux$flow.z <- Aux$flow.x - Aux$flow.y
Aux$flow.z[which(Aux$flow.z  < 0) ] <- 0

Aux$flow_sum <- Aux$flow.y + Aux$flow.z
Aux$T1 <- round((Aux$flow.y/Aux$flow_sum)*100)
Aux$T2 <- round((Aux$flow.z/Aux$flow_sum)*100)

Aux$Main <- Aux$MAIN <- 66785
Aux$Main2 <- 14487
Aux$Out <- 66881
```

```
input_turns <- left_join(input_turns, select(Aux, time, Main, Main2, T1, MAIN
    , Out, T2), by="time")


write.table(input_turns, "02_Data/traffic_state_data_turns_19_march.csv",
    quote=FALSE, sep=",", row.names=FALSE, col.names = FALSE)
```

D.4 Observed Flow and Speed Writer

This code writes the flow and speed data fin a specific format for *MATLAB*.

```
#####################################
#               TFM 2016            #
#     Code: writing_trueData.R      #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)

source("01_Codes/reading_data_RADAR.R")

##### Input for the Real Data State ####

DAY <- as.Date("2015-03-19")
Hmin <- "10:30:00"
Hmax <- "11:30:00"

## flow ###

min_H <- as.POSIXct(paste0(DAY, " ", Hmin))
max_H <- as.POSIXct(paste0(DAY, " ", Hmax))

D5 <- D4 %>%
  filter(DateTime >= min_H & DateTime < max_H) %>%
  left_join(select(loc, id, Aim_ID), by = "id") %>%
  data.frame()

Aux_int <- unique(D5[,c("Hour", "Minute")])
Aux_int <- Aux_int[order(Aux_int$Hour, Aux_int$Minute),]
Aux_int$Int <- 1:nrow(Aux_int)

D5 <- D5 %>% left_join(Aux_int, by = c("Hour", "Minute"))

D5 <- D5[order(D5$Aim_ID, D5$Int),]

D5$flow <- round(D5$flow)
D5$speed <- round(D5$speed, 4)

D6 <- select(D5, Aim_ID, Int, flow, speed)

write.table(D6, "05_Models/Calibration/scenarioInfo/trueData.txt", sep="\t",
    quote = FALSE, row.names = FALSE, col.names = FALSE)
```

D.5 Observed Travel Times Writer

This code writes the travel times data fin a specific format for *MATLAB*.

```
####################################
#               TFM 2016           #
#     Code: writing_trueTimes.R    #
#     Author: Xavier Ros Roca      #
####################################


library(dplyr)
library(stringr)

source("01_Codes/reading_data_BT.R")

##### Input for the Real Data State ####

DAY <- as.Date("2015-03-19")
Hmin <- "10:30:00"
Hmax <- "11:30:00"


## flow ###

min_H <- as.POSIXct(paste0(DAY, " ", Hmin))
max_H <- as.POSIXct(paste0(DAY, " ", Hmax))

D5 <- BT_df2 %>%
  filter(DateTime >= min_H & DateTime < max_H) %>%
  left_join(select(loc, id, Aim_ID), by = c("Start_ID" = "id")) %>%
  rename(Start = Aim_ID) %>%
  left_join(select(loc, id, Aim_ID), by = c("End_ID" = "id")) %>%
  rename(End = Aim_ID) %>%
  data.frame()

Aux_int <- unique(D5[,c("Hour", "Minute")])
Aux_int <- Aux_int[order(Aux_int$Hour, Aux_int$Minute),]
Aux_int$Int <- 1:nrow(Aux_int)

D5 <- D5 %>% left_join(Aux_int, by = c("Hour", "Minute"))


D6 <- select(D5, Start, End, Int, SampleCount, MinMeasuredTime,
    MaxMeasuredTime, AvgMeasuredTime, MedianMeasuredTime)
D6 <- D6[-which(D6$Start == 68129 & D6$End == 68130),]
D6 <- D6[-which(D6$Start == 68130 & D6$End == 68131),]

D6 <- D6[order(D6$Start, D6$End, D6$Int),]


write.table(D6, "05_Models/Calibration/scenarioInfo/trueTimes.txt", sep="\t",
     quote = FALSE, row.names = FALSE, col.names = FALSE)
```

D.6 Comparison Tables for Visualization

This code combines simulated and real data in order to be compared in a *Tableau* visualization.

```
#####################################
#               TFM 2016            #
#     Code: comparing_sim_real.R    #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)
library(RSQLite)


con <- dbConnect(drv=RSQLite::SQLite(), dbname='05_Models/Calibration/Model_
    traffic_state_21_march_prova1.sqlite')


loc <- read.csv("place.csv", sep=";", header=T, dec=",", stringsAsFactors =
    FALSE)


#### trueData (speed and flow)

trueData <- read.table("05_Models/Calibration/scenarioInfo/trueData.txt", sep
    ="\t")
names(trueData) <- c("Aim_ID", "Int", "flow", "speed")

sqlQuery <- 'SELECT oid + 0.0 as oid,  ent +0.0 as ent, flow + 0.0 as flow,
    speed + 0.0 as speed FROM MIDETEC WHERE did = 10 and sid = 1 and ent <>
    0 order by oid, ent;';
simData <- dbGetQuery(con, sqlQuery)

simData <- simData[which(simData$oid %in% loc[which(loc$Type == "R"),]$Aim_ID
    ),]

names(simData) <- names(trueData)

DF <- trueData %>%
  left_join(simData, by=c("Aim_ID", "Int")) %>%
  rename(flow_real = flow.x,
         flow_sim = flow.y,
         speed_real = speed.x,
         speed_sim = speed.y)


######


library(dplyr)
DF <- left_join(DF, select(loc, Aim_ID, id))

DF$COR_speed <- cor(DF$speed_real, DF$speed_sim)
DF$COR_flow <- cor(DF$flow_sim, DF$flow_real)

write.table(DF,  "comparing_flows.csv", row.names = FALSE, sep=";", dec=",",
    quote=FALSE )
```

```
trueTimes <- read.table("05_Models/Calibration/scenarioInfo/trueTimes.txt",
    sep="\t")
names(trueTimes) <- c("Start",
                      "End",
                      "Int",
                      "SampleCount",
                      "MinMeasuredTime",
                      "MaxMeasuredTime",
                      "AvgMeasuredTime",
                      "MedianMeasuredTime")

BT <- read.table("05_Models/Calibration/scenarioInfo/BT.txt")

simTimes_all <- data.frame(start_point = NA, end_point = NA, idveh=NA, time_
    ini=NA, time_fin=NA, ttime=NA, Int=NA)
for (i in 1:nrow(BT)) {
  sqlQuery <- paste0('select A.oid + 0.0 as start_point, B.oid + 0.0 as end_
    point, A.idveh + 0.0 as idveh, A.timedet + 0.0 as time_ini, B.timedet +
    0.0 as time_fin, B.timedet - A.timedet as ttime from (select * from
    DETEQUIPVEH where oid = ',
                     BT[i,1],
                     ' and did = 10) A inner join (select * from DETEQUIPVEH
    where oid = ',
                     BT[i,2],
                     ' and did = 10) B on A.idveh = b.idveh')
  simTimes <- dbGetQuery(con, sqlQuery)


  simTimes$Int <- ifelse(simTimes$time_ini < 38100, 1,
                         ifelse(simTimes$time_ini < 38400, 2,
                                ifelse(simTimes$time_ini < 38700, 3,
                                       ifelse(simTimes$time_ini < 39000, 4,
                                              ifelse(simTimes$time_ini <
    39300, 5,
                                                     ifelse(simTimes$time_ini
     < 39600, 6,
                                                            ifelse(simTimes$
    time_ini < 39900, 7,
                                                                   ifelse(
    simTimes$time_ini < 40200, 8,

    ifelse(simTimes$time_ini < 40500, 9,

        ifelse(simTimes$time_ini < 40800, 10,

               ifelse(simTimes$time_ini < 41100, 11, 12)))))))))))

  simTimes_all <- rbind(simTimes_all, simTimes)

}

simTimes_all <- simTimes_all[-1,]

simTimes_DT <- simTimes_all %>%
  group_by(start_point, end_point, Int) %>%
  summarise(N = length(start_point),
            Min_ttime = min(ttime, na.rm=T),
            Max_ttime = max(ttime, na.rm=T),
            Avg_ttime = mean(ttime, na.rm=T),
```

```
            Median_ttime = median(ttime, na.rm=T)) %>%
  data.frame()


simTimes <- simTimes_DT[order(simTimes_DT$start_point, simTimes_DT$end_point,
     simTimes_DT$Int),]

names(simTimes) <- names(trueTimes)

left_join(select(trueTimes, Start, End, Int, AvgMeasuredTime),
          select(simTimes, Start, End, Int, AvgMeasuredTime),
          by = c("Start", "End", "Int"))

DF_times <- trueTimes %>%
  select(Start, End, Int, AvgMeasuredTime) %>%
  left_join(select(simTimes, Start, End, Int, AvgMeasuredTime), by = c("Start
    ", "End", "Int")) %>%
  rename(Real_ttime = AvgMeasuredTime.x, Sim_ttime = AvgMeasuredTime.y) %>%
  data.frame()

DF_times <- DF_times %>%
  left_join(select(loc, Aim_ID, id), by = c("Start" = "Aim_ID")) %>%
  rename(id_start = id) %>%
  left_join(select(loc, Aim_ID, id), by = c("End" = "Aim_ID")) %>%
  rename(id_end = id)

DF_times$COR_time <- cor(DF_times$Real_ttime, DF_times$Sim_ttime)

write.table(DF_times, "comparing_times.csv", row.names = FALSE, sep=";", dec=
    ",", quote=FALSE)
```

D.7 SPSA Performance Visualization

This code prepares SPSA outputs to be visualized in *Tableau*.

```
#####################################
#               TFM 2016            #
#     Code: plots_post_sim.R        #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)

loc_path <- "05_Models/Calibration/"


A <- (read.table(paste0(loc_path, "ofval.txt"), header=F))
colnames(A) <- "ofval"
A$N <- 1:nrow(A)


B <- read.table(paste0(loc_path, "R.txt"), header=F)
colnames(B) <- paste0("corr_", c("flow", "speed", "time"))
B$N <- 1:nrow(B)
```

```
TRmin = c(85, 2, 0.7, 0.2, 0.6, 0.3, 0.5, 0.7, 2, 1, 0.2, 0.2)
TRmax = c(120, 10, 1.4, 1.0, 2.0, 1.0, 1.8, 2.2, 10, 4, 1.5, 0.8)

TR <- data.frame(cbind(TRmin, TRmax))
EXP <- expand.grid(N = 1:nrow(A), NT =1:nrow(TR))

TR <- TR %>% mutate(NT = 1:nrow(TR)) %>%
  right_join(EXP, by="NT")

C <- read.table(paste0(loc_path, "P.txt"), header=F)
params <- c("speed M",
            "speed Std",
            "Acceptance M",
            "Acceptance Std",
            "clearance M",
            "clearance Std",
            "ReactionTime",
            "ReactionTimeStop",
            "Margin for Overt. M",
            "Margin for Overt. Std",
            "Gap M",
            "Gap Std")

colnames(C) <- params
C$N <- 1:(nrow(C))


D <- C

for(i in 1:12) {
  for(j in 1:nrow(D)) {
    D[j,i] <- 10*(D[j,i]-TRmin[i])/(TRmax[i]-TRmin[i])
  }

}
names(D) <- paste0("D_", names(C))
Past <- c(102.173295,
        8.540214,
        0.688124,
        0.329139,
        1.368162,
        0.473820,
        1.077367,
        1.132899,
        5.529925,
        3.490986,
        0.583857,
        0.246785)

PastN <- Past
for(i in 1:12) {
    PastN[i] <- 10*(Past[i]-TRmin[i])/(TRmax[i]-TRmin[i])
}


E <- D[,1:12]

for(j in 1:nrow(D)) {

  for(i in 1:12) {
```

```
    E[j,i] <- D[j,i]-PastN[i]
  }

}
names(E) <- paste0("E_", names(C))

E$norm <- apply(E, 1, function(x) {norm(x, type="2")})

E <- cbind(E$norm, N = 1:nrow(E))




####
write.table(A, "A.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(B, "B.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(C, "C.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(D, "D.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(D, "D.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(E, "E.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
write.table(TR, "TR.csv", row.names = FALSE, sep=";", dec=",", quote=FALSE)
```

D.8 Goodness-of-fit Measures

This code computes different Goodness-of-fit measures for the results obtained.

```
#####################################
#               TFM 2016            #
#     Code: measuring_goodness.R    #
#     Author: Xavier Ros Roca       #
#####################################


library(dplyr)
library(stringr)


comparing_times <- read.table("V/comparing_times.csv", sep=";", dec=",",
    header=T)
comparing_flows <- read.table("V/comparing_flows.csv", sep=";", dec=",",
    header=T)


A <- matrix(ncol=6, nrow=4)
colnames(A) <- c("NRMSE", "R_squared", "U", "UM", "US", "UC")
rownames(A) <- c("Flow", "Speed", "Travel Times", "Mean")

#######

NRMSE <- function(x,y){
  n <- length(x)
  NRSME <- sqrt(sum((x-y)^2)/n)/(max(y)-min(y))
  return(NRSME)
}

U <- function(x,y) {
  n <- length(x)
```

```
  sqrt(sum((y-x)^2)/n)/(sqrt(sum(y^2)/n)+sqrt(sum(x^2)/n))
}

UM <- function(x,y) {
  n <- length(x)
  UM <- n*(mean(y)-mean(x))^2/sum((y-x)^2)
  return(UM)
}


US <- function(x,y) {
  n <- length(x)
  US <- n*(sd(y)-sd(x))^2/sum((y-x)^2)
  return(US)
}


UC <- function(x,y) {
  n <- length(x)
  rho <- cor(y,x)
  UC <- 2*(1-rho)*n*sd(y)*sd(x)/sum((y-x)^2)
  return(UC)
}



#### FLOW


Lm2 <- lm(flow_sim ~ flow_real ,comparing_flows )
SLm2 <- summary(Lm2)

A[1,1] <- NRMSE(comparing_flows$flow_sim, comparing_flows$flow_real)
A[1,2] <- SLm2$r.squared
A[1,3] <- U(comparing_flows$flow_sim, comparing_flows$flow_real)
A[1,4] <- UM(comparing_flows$flow_sim, comparing_flows$flow_real)
A[1,5] <- US(comparing_flows$flow_sim, comparing_flows$flow_real)
A[1,6] <- UC(comparing_flows$flow_sim, comparing_flows$flow_real)


#### SPEED

Lm3 <- lm(speed_sim ~ speed_real ,comparing_flows )
SLm3 <- summary(Lm3)


A[2,1] <- NRMSE(comparing_flows$speed_sim, comparing_flows$speed_real)
A[2,2] <-SLm3$r.squared
A[2,3] <- U(comparing_flows$speed_sim, comparing_flows$speed_real)
A[2,4] <- UM(comparing_flows$speed_sim, comparing_flows$speed_real)
A[2,5] <- US(comparing_flows$speed_sim, comparing_flows$speed_real)
A[2,6] <- UC(comparing_flows$speed_sim, comparing_flows$speed_real)



### TRAVEL TIME

Lm1 <- lm(Sim_ttime ~ Real_ttime ,comparing_times )
SLm1 <- summary(Lm1)

A[3,1] <- NRMSE(comparing_times$Sim_ttime, comparing_times$Real_ttime)
```

```
A[3,2] <- SLm1$r.squared
A[3,3] <- U(comparing_times$Sim_ttime, comparing_times$Real_ttime)
A[3,4] <- UM(comparing_times$Sim_ttime, comparing_times$Real_ttime)
A[3,5] <- US(comparing_times$Sim_ttime, comparing_times$Real_ttime)
A[3,6] <- UC(comparing_times$Sim_ttime, comparing_times$Real_ttime)


### mean

A[4,] <- colMeans(A[-4,])
```

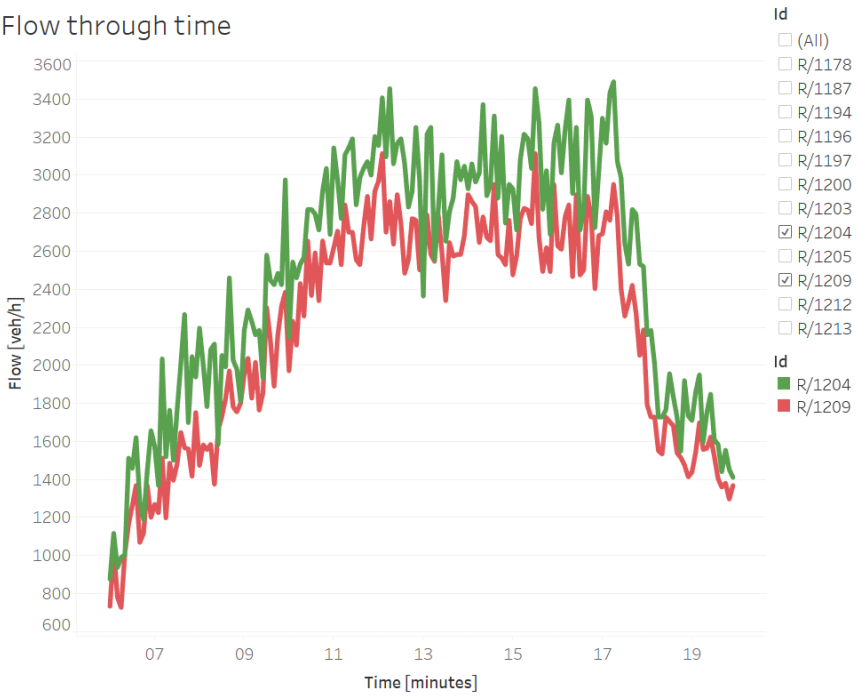
E
Tableau: Data Visualization

Tableau is a software expertized in Data Visualization. It can prepare dynamic graphics, very useful to analyze data and results. A *Tableau Public* profile have been done in order to show the dynamic results. All them can be found in:

`https://public.tableau.com/profile/xavier.ros.roca#!/`

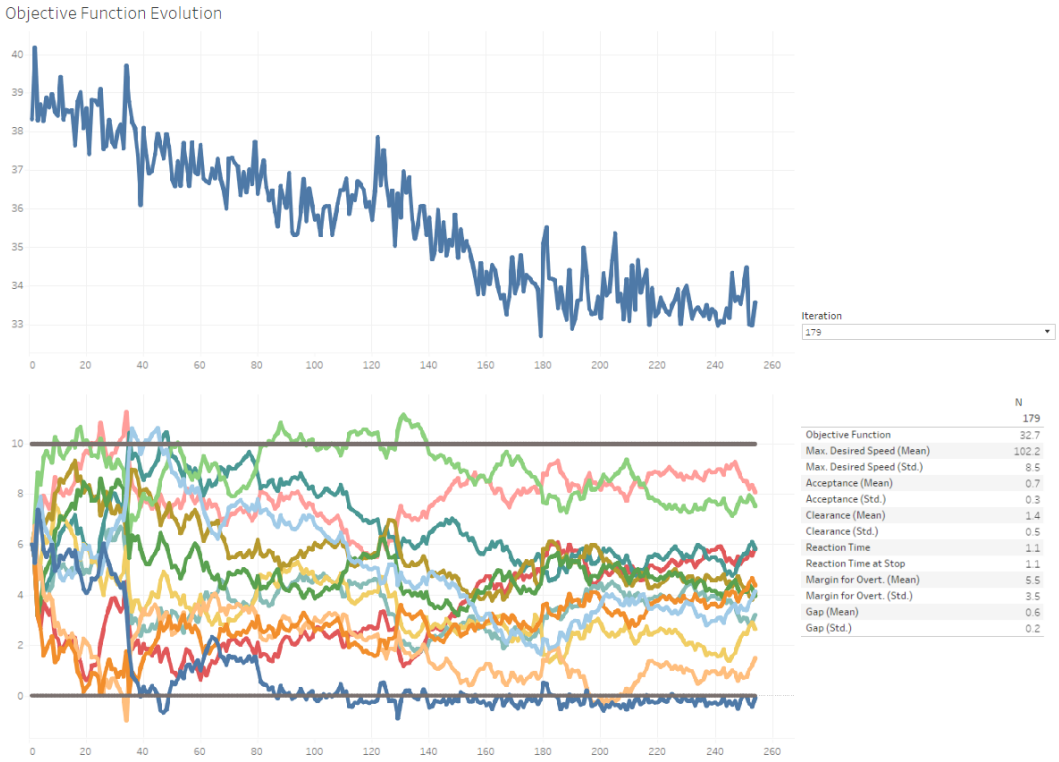
E.1 Radar data visualization

This dashboard show the real observed Data from the Radar Network. This was very useful to derive some conclusions about the position of the sensors. There are four dashboards.



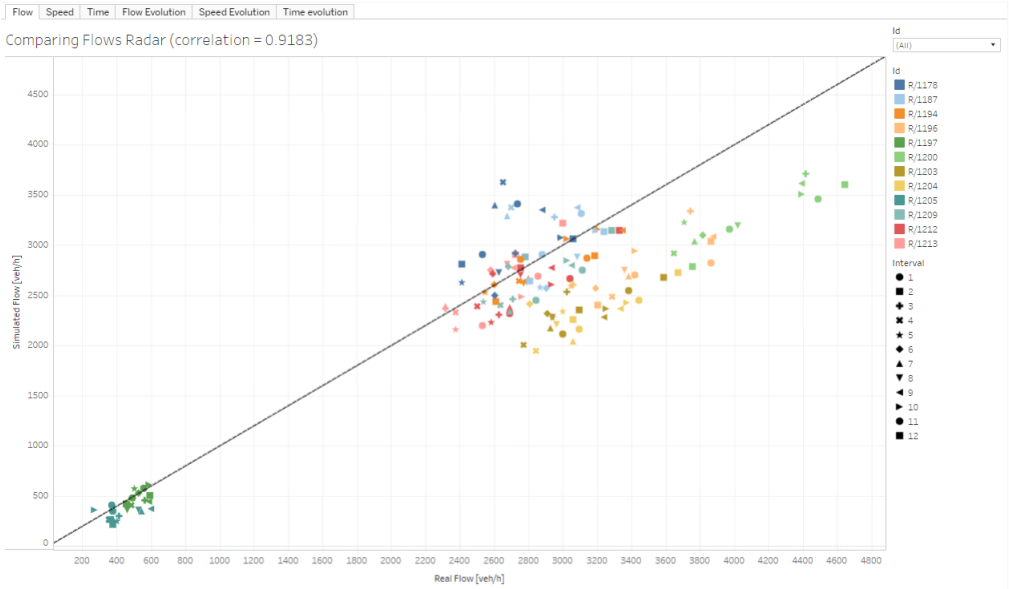
E.2 SPSA performance visualization

This dashboard shows the SPSA performance and one can consult the values for the different parameters at each iteration. There are 3 dashboards.



E.3 Calibration Results visualization

These 6 dashboards permit to analyze the relation between observed and simulated data for the Calibration Sample.



E.4 Validation Results visualization

These 6 dashboards permit to analyze the relation between observed and simulated data for the Validation Sample.